Lehrstuhl für Logik und Verifikation
Fakultät für Informatik
Technische Universität München

Asymptotic Reasoning in a Proof Assistant

Manuel Eberl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 21.09.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 01.12.2020 angenommen.

Abstract

This dissertation describes my work in the formalisation of mathematics with the proof assistant *Isabelle/HOL*, particularly mathematics related to asymptotic concepts such as limits and function growth. This work consists of both the formalisation of fundamental mathematical material for the Isabelle/HOL standard library and the creation of new proof automation tools. Both of these have since been used successfully in many formalisation applications both in pure mathematics (such as analysis and number theory) and computer science (i.e. algorithm analysis).

Specifically, this thesis presents four major contributions, each corresponding to one peer-reviewed publication under the umbrella of 'asymptotic reasoning in Isabelle/HOL':

The first of these introduces a proof automation tool that employs techniques from computer algebra to compute and prove limits and other asymptotic properties for a large class of real-valued functions (much like systems such as *Maple* or *Mathematica*). This tool has been instrumental in all my subsequent formalisation work, and no comparable instrumentation exists in other proof assistants.

Second, a formal proof of a 'cookbook method' theorem due to Akra and Bazzi, which is a sweeping generalisation of the well-known Master Theorem of divide-and-conquer recurrences. In computer science, this theorem is a staple of the analysis of divide-and-conquer algorithms. The formalisation also provides tooling to facilitate the application of the theorem to specific algorithms.

Third, a formalisation of the theory of linear recurrences and rational generating functions – in particular how to obtain closed-form solutions and asymptotic estimates of linear recurrences. These have applications in combinatorics and in the analysis of algorithms and data structures. In addition to the theorems, a verified executable solver for linear recurrences and a certifier for their asymptotics are also provided.

Finally, in order to demonstrate the usability of the aforementioned library and machinery, the last publication then describes its application to the formalisation of the vast majority of Apostol's classic textbook on analytic number theory, including the Prime Number Theorem, Dirichlet's Theorem, and many more results on the distribution of prime numbers and the asymptotic behaviour of number-theoretic functions.

Zusammenfassung

Diese Dissertation beschreibt meine Arbeit in der Formalisierung von Mathematik mit dem Beweisassistenten *Isabelle/HOL*, insbesondere von Mathematik im Zusammenhang mit asymptotischen Konzepten wie Grenzwerten und Funktionswachstum. Diese Arbeit beinhaltet sowohl die Formalisierung grundlegenden mathematischen Materials für die Isabelle/HOL-Standardbibliothek als auch die Etablierung neuer Werkzeuge zur Beweisautomatisierung. Beides wurde seitdem erfolgreich in vielen Anwendungen eingesetzt, sowohl Formalisierungen in der reinen Mathematik (wie Analysis und Zahlentheorie) als auch in der Informatik (z. B. Algorithmenanalyse).

Konkret besteht diese Arbeit aus vier Teilen im Bereich „asymptotische Beweisführung in Isabelle/HOL“ mit jeweils einer zugehörigen Veröffentlichung:

Der erste Teil ist ein Werkzeug zur Beweisautomatisierung, das Techniken aus der Computeralgebra verwendet, um Grenzwerte und andere asymptotische Eigenschaften für eine große Klasse von reellen Funktionen zu berechen und zu beweisen (ähnlich wie Systeme wie *Maple* oder *Mathematica*).

Der zweite Teil beschreibt den formalen Beweis eines kochrezeptartigen Theorems von Akra und Bazzi, das eine Verallgemeinerung des bekannten Master-Theorems für „divide-and-conquer“-Rekurrenzen darstellt. Dieses Theorem ist ein grundlegender Bestandteil der Analyse von „divide-and-conquer“-Algorithmen in der Informatik. Die Formalisierung beinhaltet außerdem Werkzeuge, die die Anwendung des Theorems auf konkrete Algorithmen angenehmer gestalten.

Der dritte Teil ist die Formalisierung der Theorie der linearen Rekurrenzen und rationalen Erzeugendenfunktionen – insbesondere wie geschlossene Lösungen und asymptotische Abschätzungen für sie gewonnen werden können. Diese finden Anwendungen in der Kombinatorik und der Analyse von Algorithmen und Datenstrukturen. Zusätzlich werden ein verifizierter ausführbarer Löser sowie eine Zertifizierer für asymptotische Abschätzungen entwickelt.

Als letzter Punkt wird, um die Nutzbarkeit der zuvor dargelegten Bibliothek und Maschinerie zu zeigen, die Formalisierung des Großteils von Apostols Lehrbuch über analytische Zahlentheorie geschildert. Dies beinhaltet formale Beweise für den Primzahlsatz, den Satz von Dirichlet, und viele andere Resultate über die Verteilung der Primzahlen und das asymptotische Verhalten zahlentheoretischer Funktionen.

Acknowledgements

This section was by far the easiest one for me to write, since there is simply so much to write about. The difficult part was really where to stop.

First of all, I would like to thank everyone who commented on or helped with any of the publications upon which this thesis is based: Jeremy Avigad, Louay Bazzi, Max Haslbeck (the elder), Johannes Hölzl, Kristina Magnussen, Tobias Nipkow, Andrei Popescu, and the numerous anonymous reviewers. I also want to additionally thank the people who commented on the thesis text itself: Mathias Fleury, Jasmin Blanchette, Max Haslbeck (the elder), Lars Hupel, Mohammad Abdulaziz, Jeremy Avigad, Uma Zalakain, Simon Wimmer, Kevin Buzzard, Julian Brunner, and Kristina Magnussen.

Next, I have to talk about the group I worked at in Munich. I had the great fortune to work in Tobias Nipkow's group as a PhD student for six years, and to be vaguely unofficially associated with it for a few years before that. I will thank a great number of people I met during that time. I tried to keep it short, failed, and decided it did not matter.

First, Peter 'The Machine' Lammich, who was the advisor of my first Bachelor's thesis and 'Praktikum'. He was probably the most athletic member of our group (which is no small feat, given the competition), and he is basically a one-man research group all by himself. His move to Brexitland was a great loss for us all.

Johannes Hölzl, the advisor of my Master's thesis, and Fabian Immler, my next-door office neighbour. They were the two mathematicians of the chair's old guard, and their departure left me as their rather inadequate replacement as resident expert for probability theory and analysis.

Julian Brunner, who introduced me to the world of competitive Super Smash Bros. Melee for the Nintendo Gamecube$^{\mathrm{TM}}$ and who served as my partner-in-crime for our highly successful electronics project (the first USB Sledgehammer) and our not-so-successful film project.

Ondřej Kunčar, who loved nothing more than to give me (and everyone else) an excuse not to work by starting lengthy discussions. I miss arguing about minutiæ of word usage in German with eight other native speakers during lunch due to one of his questions.

Mohammad Abdulaziz, always in the mood for a lengthy Socratic dialogue where he would continually ask me things about a topic I thought I knew something about until I felt like I absolutely did not. He was always easily convinced to join into whatever activity I proposed, be it bouldering, Super Smash Bros., or climbing Arthur's Seat.

Lars Noschinski, who was in the last year of his PhD when I was in my first. His suffering served as a chilling omen of what it would be like to write up a thesis.

Jonas Rädle, Kevin Kappelmann, and Lukas Stevens, the *Nesthäkchen* who managed to put together a very respectable Haskell lecture between the three of them and graciously allowed me to serve as *Master of Competition (Sr).*

Jasmin Blanchette, an amazing polyglot (certainly much more so than I!) and never-ending fountain of productivity (likewise). He was also the original *Master of Competition* and, having inherited the position from him, I can only hope I did my job half as well as he did.

Dmitriy Traytel – a pub quiz enthusiast (like me) and a PhD speedrunner (unlike me). His work on corecursion solved some of my problems years before I knew I had them. I will not pretend I understand any of it, but fortunately it was easy enough to use.

Max Haslbeck (the elder), my climbing instructor, who continuously pushes me to try harder problems. Even outside the gym, he is always full of suggestions and enthusiasm.

Simon Wimmer, aspiring ecoterrorist and fellow *Gymnasium Dingolfing* alumnus. Of all the members of our bouldering group, he was the one closest to me in skill, so it was always great fun to send with him.

Helma Piller (and before her Eleni Nikolaou-Weiss), who somehow keeps this unruly lot of us sufficiently organised. Without her, we would probably all have long since been bankrupted by unreimbursed travel expenses.

Finally, m'colleague, Lars Hupel. Fellow connoisseur of good dogs and bad puns. I very much miss his absurd enthusiasm for trains, electoral systems, and bureaucracy, and his morbid interest in looking at a piece of software and finding the most bizarre and disgusting thing he can get it to do. I also *really* miss him getting the water for our Earl Grey. I mean, that tea kitchen is *definitely* too far away.

To prevent this section from exploding even more, I will only briefly mention a number of other people who helped or influenced me in some way, and/or from whose prior work I have profited greatly: Florian Brandl, Felix Brandt, Amine Chaieb, Jacques Fleuriot, Christian Geist, Florian Haftmann, John Harrison, Joris van der Hoeven, Brian Huffman, Gregor Kemper, Angeliki Koutsoukou-Argyraki, Alexander Krauss, Wenda Li, Larry Paulson, Andrei Popescu, Bruno Salvy, René Thiemann, Thomas Türk, Akihisa Yamada, Bohua Zhan. I would also especially like to thank Larry Paulson for hosting me in Cambridge for five very productive weeks, and the CeDoSIA graduate school for paying for it.

Of course, I also want to profusely thank my advisor Tobias Nipkow. He introduced me to the world of Interactive Theorem Proving back in 2011 and supported me all the way until now in 2020, always giving me great freedom to pursue my own interests, perhaps with the occasional little nudge in the right direction. He has cultivated an exceptional group of people in Munich and I could not have found a better place to work in.

Lastly, I would also like to thank my girlfriend of many years, Kristina Magnussen, and my parents and my grandmother for their support during all those years.

Contents

1 Introduction

“[The Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine.”

— Ada Lovelace, *Sketch of the Analytical Engine* (1842)

“Wir sehen seit langem, daß diese Maschinen sich zu mehr eignen als zur planvollen Verarbeitung von Zahlen, vielmehr zu jeder in Regeln faßbaren Behandlung von Informationen irgendwelcher Art.

We have long been able to see that these machines lend themselves to more than the systematic processing of numbers, but rather to handling information of any kind in any way that can be expressed in rules.”

— Eike Jessen, *Die elektronische Rechenmaschine und unsere Gesellschaft* (1969)

At the beginning of the 20th century, there was a fierce debate among mathematicians: can mathematics be made fully formal, i.e. expressed using only symbols and a few reasoning rules? Or are there some parts of it that transcend formalism? Some, such as Hilbert, strongly believed that formalising mathematics was not only possible, but necessary. All of mathematics should be put on formally rigorous and provably consistent foundations. It was even imagined that one day one might be able to construct a machine that could eventually decide the validity of *any* mathematical statement at the push of a button.

Unfortunately, these hopes were squashed by Gödel’s groundbreaking incompleteness theorems in 1931, which, arguably, imply that there can be no such unified formal framework for mathematics. Informally, the statements of these two theorems are that any sufficiently powerful and consistent mathematical framework will have two flaws:

- There are true statements for which no proof exists.
- The consistency of the framework cannot be proven within itself.

Some years later, Turing showed a related result: the undecidability of the *halting problem*, i.e. that there cannot be a computer program which, given another computer program P as input, decides whether P terminates or not. This also implies that the ‘magic button machine’ cannot exist: if it did, we could simply encode ‘P terminates’ as a mathematical statement, push the button, and return that result.[1]

[1]Turing’s result in fact implies a variant of Gödel’s first theorem: if all true statements could be proven, then for each program P there would be a proof for ‘P terminates’ or ‘P does not terminate’, and we could solve the halting problem by searching the space of all proofs until we find a proof for either of the two statements.

However, this was *not* the end of formal mathematics: one could still hope to find a formal framework that works well in practice, i.e. we just hope that whatever statement we currently care about will turn out to be provable. In 1901, Russell's paradox ('the set of all sets that do not contain themselves') seemed to cast doubt on this, since it showed that the naïve set theory that was used at the time was fundamentally flawed as a logical system. Some believed that this only showed that formal mathematics was a fool's errand to begin with, but various logicians worked on solutions to this in the following years that have since turned out to be quite robust. Russell himself developed a new system based on the notion of *types* with the intention of preventing the kind of self-referentiality that led to the paradox he had discovered.

Based on this, Whitehead and Russell set it upon themselves to create the first fully formal and rigorous collection of mathematics: the *Principia Mathematica.* It is an enormous three-volume work, over ten years in the making and with over two thousand pages combined. In it, they attempted to develop mathematics *absolutely* formally from the ground up with a level of rigour that had never been seen before, at a time when symbolic logic was still in its infancy. They began with set theory, basic arithmetic, and real numbers, but having reached the end of the third volume in 1913, they admitted to being 'intellectually exhausted' and gave up on their plans on a fourth volume on geometry. The *Principia*'s lengthy proof leading up to the fact that $1 + 1 = 2$ is frequently used to poke fun at formal mathematics, and the book is certainly almost impossible to read. Fittingly, the historian Peter Watson called it 'one of Russell's most original publications' but also 'one of the least read of modern times.' Nevertheless, they had achieved one important goal: by the time they had written it, it was widely accepted that there were *probably* no fundamental obstacles to formalising all of known mathematics.

Although it had now been shown that formalising mathematics was possible *in principle*, it seemed completely infeasible to do so for any significant amount of non-trivial material. However, only a few years later, a real game changer appeared on the scene: the electronic computer. With it came the possibility of outsourcing the tedious and error-prone task of writing and checking formal proofs to a machine. And indeed, in 1956, Newell et al. wrote the computer program *Logic Theorist*, the first *automated theorem prover.* It was able to find proofs for 38 of the first 52 theorems from the *Principia*, and in one instance the proof it found was more elegant and satisfying than the one in the *Principia.* In a letter to Simon (one of the authors of Logic Theorist), Russell commented:

> 'I am delighted to know that *Principia Mathematica* can now be done by machinery [...] I am quite willing to believe that everything in deductive logic can be done by machinery.'

McCorduck expressed this in more philosophically coloured words in her book 'Machines Who Think' [77], stating that the success of Logic Theorist was 'proof positive [that] a machine could perform tasks heretofore considered intelligent, creative, and uniquely human'.

While these *automated theorem provers* (ATP) are without doubt a thriving and important area of research, they are not the topic of my work, and they do have their limitations: we are still very far away from giving a difficult (or even open) problem in mathematics to a computer and having any hope of it finding a proof in our lifetime. There has been one notable case where an open mathematical problem (namely the Robbins conjecture in Boolean algebra) was solved by an automated theorem prover called EQP in 1996 [78]. At the time of writing this dissertation almost 25 years later, ATPs are sometimes used to resolve open research problems in mathematics (see e.g. the work by Heule et al. [61]), but this usually requires a large amount of work *outside* the system in order to reduce it to something that the ATP understands. It seems that for now, the direct formalisation of mathematics at large – algebra, analysis, topology, etc. – is beyond the scope of ATPs.

The *Automath* [80] system by de Bruijn (started in 1967) took a different approach: the user still had to write the formal proof, but a proof checker would then verify the correctness of that proof fully automatically. In his PhD thesis in 1976, van Benthem Jutting [96] translated Landau's *Grundlagen der Analysis* [71] to Automath. The Automath approach naturally leads to the concept of *Interactive Theorem Proving* (ITP), or *Proof Assistants*: the computer does not only check the correctness of the formal proof after it has been written by a human but also while it is still being written, and it assists the human in constructing it.

This led to a large and growing family of proof assistants in the following two decades – to name just a few notable examples: NQTHM [14] (1971), Mizar [79] (1973), LCF [47] (1973), Isabelle [81] (1986), HOL [46] (1988), Coq [24] (1989). All of these (or in some cases their numerous descendants) are still in use at the time of writing this thesis (2020). For some of these systems (especially in the HOL and NQTHM family) this development was particularly spurred on by industrial applications in the verification of software and hardware, but it seems that most or even all of them have been used at least partially for the formalisation of mathematics as well.

Indeed, as the capabilities of the systems and the sizes of their communities grew, the formalisation of mathematics has made great strides since the early 2000s. To illustrate this, consider the following list of groundbreaking and successful formalisation projects in mathematics:

- Prime Number Theorem, elementary Selberg–Erdős proof (Isabelle/HOL, 2004) [7]
- Four Colour Theorem (Coq, 2005) [44]
- Jordan Curve Theorem (HOL Light, 2007) [54]
- Prime Number Theorem, complex-analytic proof (HOL Light, 2009) [58]
- Feit–Thompson Theorem (Coq, 2012) [45]
- Kepler's Conjecture (HOL Light and Isabelle/HOL, 2015) [55]
- Chaoticity of the Lorenz Attractor (Isabelle/HOL, 2018) [65]
- Independence of the Continuum Hypothesis (Lean, 2019) [56]

This flurry of activity was aided by improvements to the systems themselves, such as growing libraries of basic mathematical definitions, facts, and theories (e.g. algebraic structures, differential equations, complex analysis) and better tools to make writing definitions and proofs easier for the user (e.g. type classes, proof automation, recursive function definitions, algebraic (co-)datatypes, structured proof languages). Work on both of these aspects will be an important part of this thesis.

The formalisation of the Kepler conjecture is particularly noteworthy for a number of reasons: first of all, the sheer number of collaborators (the arXiv paper on the formalisation lists 22 authors); second, the conjecture had only been resolved a few years before formalisation began. The proof by Hales relied heavily on computations done by computer programs written specifically for the proof and the referees found themselves overwhelmed by the task of checking the correctness of these programs. This led Hales – a mathematician with no prior association with the theorem-proving community – to spearhead the formalisation effort himself. This illustrates a first and very important reason why formalisation of mathematics might be interesting both to mathematicians and to society in general: it allows us to convince ourselves beyond reasonable doubt that a mathematical proof is correct when the traditional refereeing process fails, such as for computer proofs.

The other projects mentioned above were also very daunting at the time. For instance, shortly before Harrison formalised the analytic proof of the Prime Number Theorem in HOL Light, Bob Solovay had predicted[2] that proof assistants would not be ready to formalise this proof for decades. This brings us to another reason to formalise mathematics: to demonstrate that *it can be done.*

The last reason is an even more frivolous one but a very important one to me nonetheless: Because it is *fun.* There is a profound satisfaction innate to formalisation with a proof assistant that can only be compared to mastering an exceptionally difficult computer game (a comparison e.g. made half-jokingly by Tobias Nipkow when introducing the *Isabelle* proof assistant in his *Semantics* lecture). The formaliser (or player) tackles one proof obligation after another, frantically working back and forth to make a connection between the assumptions and the goal, backtracking and trying another route when entering a dead end, and rejoicing upon finally reaching 'No subgoals!'. While this may sound just like 'pen-and-paper' mathematics, the difference is that the computer, ever vigilant and pedantic, gives immediate feedback at every single step and does not allow even the slightest vagueness in an argument. There is hardly ever any uncertainty about whether what one has written down is correct or not. Thus, the feedback loop of punishment and reward is very fast – just like in a computer game.

I discovered this computer game in my first year as an undergraduate in 2011 in the aforementioned lecture by Tobias Nipkow. The ability to formalise arbitrary statements from computer science and mathematics at this level of clarity immediately impressed and fascinated me greatly.[3] It is no exaggeration to say that this experience completely derailed the course of my university studies – but unlike other highly addictive computer games, it was probably not to the detriment of my academic career. From that point onward, it was interactive theorem proving all the way for me, and that has now led me to this PhD project.

[2]See the paragraph on the Prime Number Theorem in Freek Wiedijk's 'The Seventeen Provers of the World' [101].

[3]Indeed, in my experience, many mathematicians still seem to be under the impression that this is not possible. A particularly common misconception is that because they have finite memory, computers can only ever check finitely many cases by exhaustion but not possibly reason about infinity and non-discrete objects like real numbers or transcendental functions. An odd thing to think, in my opinion, considering they seem to be able to do it just fine with a finite brain and a finite piece of paper in front of them. Perhaps this is a remnant of the aforementioned idea that mathematical reasoning is something 'intelligent, creative, and uniquely human.'

In this publication-based thesis, I will showcase four particular steps from one important part of my interactive theorem proving journey. These correspond to four peer-reviewed articles of mine that can be found in the appendix, bound together by the common theme of *asymptotics*: the study of limits and the growth of (mostly real-valued) functions, a topic that had previously not been given enough attention in our community.

2 Outline

As stated before, the thesis is centred around four peer-reviewed publications, which can be found in the appendix. Most of the remainder of the thesis serves to provide summaries, background information, and context for these.

Chapter 3 gives some general background required to understand the work that is being presented: Section 3.1 presents a brief overview of the *Isabelle* theorem prover, followed by a more detailed explanation of how interactive theorem proving is done in Isabelle in practice in Section 3.2. Section 3.3 explains the *visible universe of Isabelle applications*, where formalised material in Isabelle is published, and how it is maintained. Lastly, Section 3.4 explains some notational conventions and terminology relevant to the rest of this work and Section 3.5 focuses in particular on the asymptotic notation known as *Landau symbols*.

Next, Chapter 4 gives a brief synopsis of the four publications as well as a short list of additional related contributions without a formal publication attached to them. Chapters 5 to 8 then each give more detailed summaries, motivation, background information, and in some cases outlook regarding one of the publications.

Finally, Chapter 9 concludes with some additional noteworthy formalisation work that I have done during my time as a PhD student and some remarks on what I consider important for the future of the formalisation of mathematics.

3 Preliminaries

“Formal verification is like opening cans of worms, and then eating them.”

— Robert Sison, via Twitter (2020)

3.1 The Isabelle Proof Assistant

For the work presented in this thesis (and all of my work in general), I used the proof assistant *Isabelle*, which was originally developed by Larry Paulson in 1986 and extended by various other people after that, including me. What sets Isabelle apart from most other proof assistants is the fact that it is *generic*: it provides a very minimalistic logic called Isabelle/Pure, which only provides very basic logical concepts such as implication and universal quantification. On top of this, various other *object logics* can be (and have been) built. At the time of writing this dissertation, the most used logic in Isabelle by a very large margin is *higher-order logic* (HOL), and this is the logic that I used as well. Isabelle's version of HOL is very similar to those of the other systems from the HOL family of proof assistants (e.g. HOL4, HOL Light, HOL Zero, Proof Power), and there is ongoing work on importing HOL4 proofs into Isabelle/HOL [66]. For the sake of completeness, I will mention two other object logics available in Isabelle. First, there is *Zermelo–Fraenkel set theory*, which is considered the usual 'default' logic for traditional mathematics (but is not very well-developed in Isabelle). Second, there is recent and very ambitious work to create support for *Homotopy Type Theory* as well [22].

Any theorem proving system that is usable for practical developments will likely consist of a fairly large code base. Consequently, steps must be taken to still ensure the soundness of the overall system, i.e. that any theorem that can be produced by the system really is a valid fact in the underlying logic. There are various approaches for this, but I will only mention the one used by Isabelle: like the systems of the HOL family, the design of Isabelle roughly follows what is known as the LCF model (named after the early experimental theorem prover of the same name). The full formal proofs (also called *proof terms* or *proof objects*) are not recorded at all. Soundness is instead ensured through a mechanism of the programming language in which the system is implemented (Standard ML). A type `thm` of theorems is defined in such a way that only a relatively small set of functions can inspect and manipulate values of this type. These functions form the *inference kernel*, and all other code can only create or modify such `thm` values by using this kernel. Therefore, in principle, only the correctness of the kernel is critical for the soundness of the resulting system. The kernel is *relatively* small and only provides very basic operations such as *modus ponens*, eliminating a universal quantifier, or making a non-recursive equational definition.

The Standard ML implementation and runtime that Isabelle is based upon is Poly/ML [76], which allows on-the-fly compilation of code. Isabelle makes use of this by providing an `ML` command with which users can add arbitrary program code to the system at any point (many other systems have similar features). This allows advanced users to create their own Isabelle commands, including proof automation, definitional tools, diagnostic and visualisation tools, or interfaces to external tools (such as automated theorem provers or computer algebra systems). Regarding user-defined proof automation tools, it should again be mentioned that due to the LCF approach, bugs in user code cannot lead to logical inconsistencies by design. Interestingly, most of Isabelle's basic commands are not 'built-in' but also constructed in this way with only a fairly minimal amount of bootstrapping. In this sense, there is no clear distinction between an Isabelle *user* and an Isabelle *developer*.

Lastly, I must mention one very important feature of Isabelle, even if it is only tangentially relevant for the work presented here: the *code generator* [52, 53]. Isabelle's code generator can automatically generate executable code from Isabelle/HOL definitions, provided that these definitions fall within a suitable 'computational' subset of HOL (e.g. no unbounded quantification or choice operators). For definitions that are not directly computational in this sense or that are too inefficient, *code equations* can be provided. These are 'alternative definitions' to be used by the code generator, but their equivalence to the original definitions must be proven by the user. In this dissertation, the only place where I make direct use of the code generator is in order to provide an executable solver and certifier for linear recurrences (see Section 7). However, the code generator is also of great indirect use: it is used by QuickCheck [17], an automatic random counterexample generator. Every time a user attempts to prove a theorem, QuickCheck tries to use the code generator to find counterexamples for the theorem statement to let the user know when they are trying to prove something that does not hold. This happens fairly frequently in interactive theorem proving due to typos, forgotten preconditions, etc.

3.2 Interactive Theorem Proving in a Nutshell

In this section, I will explain what using Isabelle looks like in practice. Some of this is applicable to other systems as well, but some aspects (such as structured proofs) are fairly specific to Isabelle. A good (although now already somewhat dated) overview of various systems and what proofs typically look like in each of them can be found in Wiedijk's book *The Seventeen Provers of the World* [101].

Definitions in a theorem prover often look much like code in a programming language – especially functional ones such as Haskell or OCaml, since many logics (including HOL) contain all the essential features of a functional programming language.

Like functional programming languages, many systems draw heavily from Church's λ calculus, including the notation $f\ x\ y$ for function application instead of the $f(x, y)$ used in traditional mathematics, and $\lambda x.\ t$ for an anonymous function $x \mapsto t$.

Typically, the user writes some definitions, theorem statements, and proofs, and then the computer checks the well-formedness and correctness of the input. In contrast to programming languages, however, the processes of writing the code and interpreting it are intertwined: the user does not write everything at once, runs the checker, repairs small mistakes, re-runs the checker, etc. but rather writes definitions and proof steps one by one, receiving immediate feedback from the system. The system provides dynamic information about the current context (e.g. what variables are in scope and what their types are, which proof obligations remain to be shown) and it would be difficult if not impossible to write a proof in the system without that information. This requires very tight integration with the editor, and the only Isabelle editor currently suitable for production use is *Isabelle/jEdit* [98].

Many proof assistants are traditionally centred around *tactics*, which are small commands that transform a proof goal into one or more new goals (or solve it completely). For instance, to prove a goal such as $\forall x.\ P\ x \wedge Q\ x$, one could apply a tactic that performs $\forall$-*introduction*, which adds a new fixed free variable x to the scope and the new goal $P\ x \wedge Q\ x$. Next, one could use $\wedge$-*introduction*, which leaves us with the two goals $P\ x$ and $Q\ x$. This is done until all goals have been solved – for non-trivial theorems, such proofs usually consist of long sequences of tactic invocations.

Isabelle/HOL provides many such tactics, including

- very low-level tactics like `rule` to apply resolution with a single theorem (as was done above) or `subst` to perform a single step of equational rewriting,
- general-purpose automation such as the *simplifier* or `auto`, which rely on a large extensible database of rewriting rules, tableaux reasoning, and other tricks, and
- special-purpose automation for e.g. Presburger arithmetic, SAT solving, or approximation of transcendental functions.

Historically, Isabelle proofs used to consist entirely of such long tactic scripts – possibly hundreds of them for larger theorems. However, such proofs are very unreadable and difficult to maintain, which is problematic considering the vast amount of material in the Isabelle universe and the fact that the underlying system and basic library has undergone and is still undergoing great changes on a regular basis.

Therefore, most modern Isabelle proofs are written in the Isar proof language introduced by Wenzel [99, 100] (which was heavily inspired by Mizar). This allows structured proofs that are much closer to the kind of reasoning mathematicians actually do on paper instead of the awkward 'backward' reasoning of tactic scripts.

To illustrate what these proofs look like in practice, let us look at a concrete example. Consider the statement $\gcd(ca, cb) = c \cdot \gcd(a, b)$. Figures 3.1 and 3.2 show a structured Isar proof and an 'old-style' tactic script proof of this statement, respectively. Figures 3.3 to 3.6 show proofs in some other popular systems, taken from their respective standard libraries.[1]

[1]Note that the Isabelle/HOL and Lean versions are for general rings and include a 'normalize' operation that is required due to the fact that the GCD is defined to be a canonical representative (e.g. $\gcd(4, 6)$ is 2 and not -2). For the other systems, the proof for the GCD on natural numbers is shown, where no normalisation is needed.

```
lemma (in semiring_gcd) gcd_mult_left: ‹gcd (c * a) (c * b) = normalize (c * gcd a b)›
proof (cases ‹c = 0›)
  case True
  then show ?thesis by simp
next
  case False
  from ‹c ≠ 0› have ‹c * gcd a b dvd gcd (c * a) (c * b)›
    by (auto intro: gcd_greatest)
  moreover from ‹c ≠ 0› have ‹gcd (c * a) (c * b) div c dvd gcd a b›
    by (intro gcd_greatest) (auto simp: div_dvd_iff_mult algebra_simps)
  from this and ‹c ≠ 0› have ‹gcd (c * a) (c * b) dvd gcd a b * c›
    by (subst (asm) div_dvd_iff_mult) auto
  ultimately have ‹normalize (gcd (c * a) (c * b)) = normalize (c * gcd a b)›
    by (auto intro: associated_eqI simp: algebra_simps)
  then show ?thesis
    by (simp add: normalize_mult)
qed
```

Figure 3.1: Structured Isabelle/HOL proof (adapted from the distribution)

```
lemma (in semiring_gcd) gcd_mult_left: ‹gcd (c * a) (c * b) = normalize (c * gcd a b)›
  apply (cases ‹c = 0›)
   apply simp
  apply (rule associated_eqI)
     apply simp
   apply (subst mult.commute, subst div_dvd_iff_mult [symmetric]; simp)
    apply (simp add: div_dvd_iff_mult mult_ac)
  apply simp_all
  done
```

Figure 3.2: Isabelle/HOL tactic script proof (adapted from the distribution)

```
let GCD_LMUL = prove(
  `!a b c. gcd(c * a, c * b) = c * gcd(a,b)`,
  REPEAT GEN_TAC THEN CONV_TAC SYM_CONV THEN
  ONCE_REWRITE_TAC[GSYM GCD_UNIQUE] THEN
  REPEAT CONJ_TAC THEN TRY(MATCH_MP_TAC DIVIDES_MUL_L) THEN
  REWRITE_TAC[GCD] THEN REPEAT STRIP_TAC THEN
  REPEAT_TCL STRIP_THM_THEN (SUBST1_TAC o SYM)
   (SPECL [`a:num`; `b:num`] BEZOUT_GCD) THEN
  REWRITE_TAC[LEFT_SUB_DISTRIB; MULT_ASSOC] THEN
  MATCH_MP_TAC DIVIDES_SUB THEN CONJ_TAC THEN
  MATCH_MP_TAC DIVIDES_RMUL THEN ASM_REWRITE_TAC[]);;
```

Figure 3.3: HOL Light proof (standard library, `Library/prime.ml`)

```
Lemma gcd_mul_mono_l :
  forall n m p, gcd (p * n) (p * m) == p * gcd n m.
Proof.
 intros n m p.
 apply gcd_unique'.
 apply mul_divide_mono_l, gcd_divide_l.
 apply mul_divide_mono_l, gcd_divide_r.
 intros q H H'.
 destruct (eq_0_gt_0_cases n) as [EQ|LT].
 rewrite EQ in *. now rewrite gcd_0_l.
 destruct (gcd_bezout_pos n m) as (a & b & EQ); trivial.
 apply divide_add_cancel_r with (p*m*b).
 now apply divide_mul_l.
 rewrite <- mul_assoc, <- mul_add_distr_l, add_comm, (mul_comm m), <- EQ.
 rewrite (mul_comm a), mul_assoc.
 now apply divide_mul_l.
Qed.
```

Figure 3.4: Coq proof, tactic style (standard library, `Numbers/Natural/Abstract/NGcd.v`)

```
Lemma muln_gcdr : right_distributive muln gcdn.
Proof.
move=> p m n; have [-> //|p_gt0] := posnP p.
elim/ltn_ind: m n => m IHm n; rewrite gcdnE [RHS]gcdnE muln_eq0 (gtn_eqF p_gt0).
by case: posnP => // m_gt0; rewrite -muln_modr //=; apply/IHm/ltn_pmod.
Qed.
```

Figure 3.5: Coq proof, *SSReflect* style (Mathematical Components, `div.v`)

```
theorem gcd_mul_left (a b c : α) : gcd (a * b) (a * c) = normalize a * gcd b c :=
classical.by_cases (by rintro rfl; simp only [zero_mul, gcd_zero_left, normalize_zero]) $
     assume ha : a ≠ 0,
suffices gcd (a * b) (a * c) = normalize (a * gcd b c),
  by simpa only [normalize_mul, normalize_gcd],
let ⟨d, eq⟩ := dvd_gcd (dvd_mul_right a b) (dvd_mul_right a c) in
gcd_eq_normalize
  (eq.symm ▸ mul_dvd_mul_left a $ show d ∣ gcd b c, from
    dvd_gcd
      ((mul_dvd_mul_iff_left ha).1 $ eq ▸ gcd_dvd_left _ _)
      ((mul_dvd_mul_iff_left ha).1 $ eq ▸ gcd_dvd_right _ _))
  (dvd_gcd
    (mul_dvd_mul_left a $ gcd_dvd_left _ _)
    (mul_dvd_mul_left a $ gcd_dvd_right _ _))
```

Figure 3.6: Lean proof (*mathlib*, `algebra.gcd_domain`)

Next, let us turn from proofs to definitions. Isabelle offers various definitional tools:

- The most basic one, the `definition` command [99], defines a new constant or non-recursive function and can be regarded as a simple abbreviation that can be unfolded. This is just a very thin wrapper around the basic definitional mechanisms provided by Isabelle's kernel.
- The `primrec` command [12] allows primitively-recursive definitions for natural numbers or algebraic datatypes.
- The `function` package [69, 70] supports more complex recursion patterns including nested and mutual recursion or recursions with non-obvious termination arguments (which must then be provided by the user).
- The `inductive` and `coinductive` commands [99] allow defining inductive and coinductive predicates, i.e. describing predicates by giving a set of inference rules.
- The `corec` command [13] allows defining corecursive functions (e.g. to transform infinite streams).
- The (co-)datatype package [12, 13] provides support for defining algebraic datatypes and codatatypes.

Again, all of these tools must go through the Isabelle kernel to do their work – every definition made with them ultimately reduces to the small logical primitives offered by the kernel.

Definitions and proofs can be organised in modules, which are referred to as *theories*. Each theory is one file, and each theory can import several other theories in order to use all the definitions and theorems from them and their respective dependencies. A typical Isabelle theory will have between several hundred and several tens of thousands of lines. One step up in the hierarchy, theories can be bundled together into a *session*. Such sessions can be *built* (i.e. processed in batch) and the resulting state can be stored on disk. This allows a user to quickly use the results from a session without having to process all of its material every time Isabelle is started (which, depending on the size of the session, can take between several minutes and several hours).

3.3 The Isabelle Distribution and the Archive of Formal Proofs

The Isabelle distribution is what a user would download in order to use Isabelle. It contains a large number of tools (including the ones mentioned in the previous section) and a multitude of *theories*, written by many different contributors over the years. The most important sessions in the context of this work are (in the 2020 release of Isabelle):

- `Pure`, which contains the bootstrapping of Isabelle's metalogic
- `HOL`, which provides the axiomatisation of the HOL object logic, all the basic tools and tactics mentioned before, the code generator, and much basic library material e.g. on natural numbers, integers, and lists, but also some more advanced mathematics (basic algebra, topology, and analysis)

- `HOL-Computational_Algebra` (primes, fundamental theorem of algebra, Euclidean domains, univariate polynomials, formal power series)
- `HOL-Number_Theory` (residue rings, Euler's φ function, primitive roots, quadratic reciprocity)
- `HOL-Analysis` (advanced linear algebra, topology, analysis, measure and integration theory)
- `HOL-Complex_Analysis` (complex analysis including e.g. Cauchy's integral formula, the residue theorem, the great Picard theorem, the Riemann mapping theorem)
- `HOL-Library` (various unsorted material such as Landau symbols, Stirling numbers, permutations, multisets)

Only a comparatively small number of core developers can directly contribute to and maintain the material in the Isabelle distribution, although there are occasional contributions from the outside (which are then integrated by a member of this core group). The distribution contains, for the most part, only fairly 'general-purpose' formalisations and tools.

For more special-purpose contributions, there is the *Archive of Formal Proofs* (AFP)[2], a collection of Isabelle formalisation projects on various topics in mathematics and computer science. At the time of writing this dissertation, it contains 539 entries from 356 authors and a total of roughly 2.5 million lines of formal proof documents, and all of these numbers have been growing steadily. All submitted entries are reviewed by a board of five editors.[3] The Isabelle community ensures that all entries always work with the most recent version of Isabelle. This archive is the most important repository for Isabelle proof developments, and when publishing an article on a formal proof development in a journal or at a conference, it is customary to include a reference to the corresponding AFP entry.

The Isabelle distribution and the AFP together form what Makarius Wenzel calls the 'visible universe of Isabelle'. Its contents are always kept up-to-date with the latest developments of Isabelle (mostly by the core developers) in a continuous maintenance effort. This is of great importance, because developments outside this universe tend to 'break' more and more unless they are actively maintained (as Isabelle continues to evolve while they stagnate). For that reason, users are greatly encouraged to submit their developments to the AFP.

3.4 Notation and Terminology

In this thesis, I will use the terms 'interactive theorem prover' and 'proof assistant' interchangeably. I will also sometimes simply speak of 'theorem provers' when only interactive ones are meant. To those among the readers who are not from the theorem proving community, I must also clarify that when I say in this thesis that I proved some statement or defined some concept, I *always* mean that I wrote a formal proof or definition for it in the Isabelle system, and usually that I was the first person who did this. In almost all circumstances, I was of course *not* the first person to work these things out on paper.

[2] `https://isa-afp.org`

[3] I myself have been an editor since 2018.

Both in the main part of the thesis and the attached articles, I mostly avoid showing Isabelle syntax in order to keep the presentation as concise and readable as possible, and to emphasise that most of what I present is not specific to Isabelle but could well be done in other system as well, given enough time and effort.

As for mathematical notation, there is not terribly much of it to introduce here that is both unusual or unclear enough to deserve explanation *and* used by more than one of the four different articles that will be presented in this thesis. Any notation relevant to only one of them will be introduced in the corresponding chapter, or in the article itself. For the most part, note only that:

- I will make use of λ notation for functions as mentioned before.
- Function application will sometimes be written in the mathematical style $f(x, y)$ and sometimes as $f\ x\ y$ as appropriate.
- log will always denote the natural logarithm.
- I will always write $\log^k x$ to denote $(\log x)^k$, *not* the iterated logarithm $\underbrace{\log \ldots \log}_{k \text{ times}} x$.

However, there is one important shared piece of notation that does require more extensive clarification, as its use in the literature is often quite imprecise, namely that of *Landau symbols*, which will be explained in great detail in the next section.

$$\begin{array}{rcl}
f \in O(g) & \longleftrightarrow & \exists c > 0.\ \exists x_0.\ \forall x \geq x_0.\ |f(x)| \leq c \cdot |g(x)| \\
f \in o(g) & \longleftrightarrow & \forall c > 0.\ \exists x_0.\ \forall x \geq x_0.\ |f(x)| \leq c \cdot |g(x)| \\
f \in \Omega(g) & \longleftrightarrow & \exists c > 0.\ \exists x_0.\ \forall x \geq x_0.\ |f(x)| \geq c \cdot |g(x)| \\
f \in \omega(g) & \longleftrightarrow & \forall c > 0.\ \exists x_0.\ \forall x \geq x_0.\ |f(x)| \geq c \cdot |g(x)| \\
f \in \Theta(g) & \longleftrightarrow & f \in O(g) \wedge f \in \Omega(g) \\
f \sim g & \longleftrightarrow & f(x) - g(x) \in o(f(x)) \quad \longleftrightarrow \quad f(x) - g(x) \in o(g(x))
\end{array}$$

Table 3.1: Definitions of the five Landau symbols and asymptotic equivalence (for $x \longrightarrow \infty$; analogously for other filters)

3.5 Landau Symbols

This notation for the asymptotic behaviour of functions (or rather some variation of it) has its origins in works by Paul Bachmann and Edmund Landau. It is sometimes also called *Bachmann–Landau notation* or simply 'Big-O' after the O symbol (which is the most commonly used Landau symbol in practice).[4] The intended meaning of e.g. $f(x) \in O(g(x))$ is that the function $f(x)$ is at most of the order of $g(x)$, i.e. $f(x)$ is bounded by some multiple of $g(x)$. The precise meaning, however, is often unclear:

- What does this mean for functions that take negative values? Is it $f(x) \leq c\,g(x)$? Or perhaps $|f(x)| \leq c\,|g(x)|$? Or even $|f(x)| \leq c\,g(x)$?
- Does the boundedness have to hold for *all* x or only for sufficiently large x? Or for x sufficiently close do 0?
- Is $g(x)$ implicitly assumed to always be non-zero? Or at least for sufficiently large x?
- If we restrict ourselves e.g. to integer-valued functions, can c also only be an integer or can it also be a real number?

In some contexts, these different conventions are not too significant. For instance, in algorithm analysis, the functions that are considered are typically non-negative, so that the first question is irrelevant. They are also often defined on the non-negative integers, and the $g(n)$ on the right-hand side is typically positive except for possibly a few exceptions, so that the second and third issue also mostly disappear.

For sufficiently well-behaved positive functions, the question of whether the boundedness holds for all x or only for sufficiently large x can be seen to be irrelevant by choosing a large enough constant c. It is therefore often convenient to use the convention that it holds only *eventually* when showing a 'Big-O' statement and to switch to the convention that it holds for *all* inputs when using a previously proven 'Big-O' statement.

[4]Knuth [68] gives a brief overview of the history of the various Landau symbols.

In algorithm analysis, Landau symbols are usually used to study the behaviour of functions for $x \longrightarrow \infty$. However, in mathematical analysis, the $O(\ldots)$ symbol is also often used in the context of the local behaviour of real or complex functions. For instance, Taylor's theorem for a function $f : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ that is k-times differentiable at x is often written as

$$f(x+h) = f(x) + f'(x)h + \ldots + \frac{f^{(k)}(x)}{k!}h^k + R(h) \quad \text{where } R(h) \in O(h^{k+1})\ ,$$

where the $O(\ldots)$ implicitly contains the restriction 'for sufficiently small h.'[5]

In the context of a theorem prover, a clear choice for the definition of $O(\ldots)$ etc. must be made once and for all. In Isabelle, the situation is now as follows: I defined five Landau symbols, plus the related notion of *asymptotic equivalence*, written as $f(x) \sim g(x)$. Their definitions can be found in Table 3.1. These symbols were originally bundled in an AFP entry called *Landau Symbols*, but were later moved to the Isabelle distribution to allow using them also to aid proofs within the standard library.

When I designed these definitions at the beginning of my PhD project, I had to attempt to choose one particular precise meaning while still allowing the user to use these symbols in a flexible way in the wide variety of different contexts in which they are practically used. Since there are five Landau symbols in total, the choices should also be uniform among all these symbols and make sense for all of them. In the end, I made the following decisions:

- No assumptions on zeroness or non-negativity of functions are made.
- The absolute value (or rather the *norm*) is used in order to make the Landau symbols truly scale-invariant in the sense that $O(af(x))$ and $O(f(x))$ are the same for any constant $a \neq 0$.
- c can always be any positive real number.
- The precise neighbourhood in which the boundedness should hold is $x \longrightarrow \infty$ by default but can be adjusted with an additional argument.

To elaborate on the last point, the Isabelle/HOL library (like some libraries in other proof assistants) uses *filters* [64], a concept going back to Bourbaki, to denote topological concepts such as neighbourhoods and limits. These filters are very powerful and modular abstractions for such topological matters (and much more), but the details of how they are defined and how they work are not relevant for this thesis. Let me only give these examples: the 'at-top' filter corresponds to $x \longrightarrow \infty$ (i.e. the neighbourhood of ∞ in a linear order), whereas the 'at c' filter corresponds to the pointed neighbourhood of the point c.

This filter-centred approach to Landau symbols (which was suggested to me by Johannes Hölzl) is very flexible. For example, the following can be achieved:

- The abovementioned bound in Taylor's theorem can be written as '$R(h) \in O_{\text{at } 0}(h^{k+1})$'.

[5]To complicate matters even more, both mathematicians and computer scientists often write terms such as $f(x) = e^{x+O(1)}$ where $O(1)$ implicitly stands for 'some function that is $O(1)$.' Enabling such notation in the context of a proof assistant is difficult, although there is work on this in Coq [1]. In some contexts, it is already possible to write such things in Isabelle, but I for one do not make use of it.

- The alternative meaning of Landau symbols where the inequality must hold for *all* inputs can be achieved by using the 'top' filter, which represents the entire set of possible inputs for the function (not to be confused with the 'at-top' filter).
- By using *product filters*, the Landau symbols can be used for multivariate functions as well. This was done e.g. by Zhan and Haslbeck [103] to analyse algorithms taking more than one input.
- Suppose we are analysing the sorting algorithm *insertion sort*, which sorts a given list x with n elements, taking $t(x)$ steps of time. It is well-known that $t(x) \in O(n^2)$ for sufficiently large n, but it is not immediately clear how to capture this formally since the left-hand side is in terms of the list x, not its size n. Clearly, the intended meaning is somehow that this holds for all lists of size n for any sufficiently large n. In Isabelle, one can actually easily achieve exactly this meaning by writing

  $$t(x) \in O_{\text{length } \textbf{going-to} \text{ at-top}}(\text{length}(x)^2)\ ,$$

  with the 'going-to' filter stating that the intended asymptotic meaning is that it is the *length* of the input that must be sufficiently large.
- Lastly, Stirling's formula for the complex-valued Γ function states that

  $$\Gamma(z) \sim \sqrt{\frac{2\pi}{z}}\left(\frac{z}{e}\right)^z$$

  uniformly as $|z| \longrightarrow \infty$ but only if z lies within the cone $|\arg(z)| \leq \alpha$ for a fixed angle $\alpha < \pi$. We can write this in Isabelle by annotating the $\sim$ with the filter

  $$\text{at-infinity} \ \sqcap \ \text{principal (complex-cone } (-\alpha)\ \alpha)$$

  where $\sqcap$ denotes the greatest lower bound of two filters (the intersection, in a sense).

This illustrates how widely usable and flexible Isabelle's Landau symbols are.

4 Summary of Contributions

The overall theme of the work presented here is the formalisation of mathematics in a proof assistant (or interactive theorem prover), with a particular focus on asymptotic properties. I used the aforementioned Isabelle/HOL as the underlying system, but the work could just as well be replicated in any proof assistant that uses some form of classical logic suitable to the formalisation of standard mathematics. In the following, I will give a very brief and high-level overview of the work presented in the four publications that form this publication-based dissertation – more detailed summaries and background can be found in the remaining chapters.

Semi-Automatic Real Asymptotics [39]. This publication presents an automatic proof procedure very much in the spirit of the series expansion procedures implemented in computer algebra systems. It facilitates reasoning about limits and other asymptotic properties of concrete real-valued functions. A large class of functions is covered (combinations of basic arithmetic, exp, log, sin, etc.).

Divide-and-Conquer Recurrences [30]. Here, I describe the first formalisation of the *Akra–Bazzi Theorem* on the asymptotics of divide-and-conquer recurrences. It is a generalisation of the well-known *Master Theorem*. These recurrences often arise in the asymptotic running time analysis of divide-and-conquer algorithms. Such algorithms (including e.g. Merge Sort and Karatsuba multiplication) and their analysis are a staple of undergraduate algorithms courses and textbooks (see e.g. Cormen et al. [25]) and are of great practical importance as well. The precise solutions to such recurrences are typically very complicated and can often not be expressed in a closed form, but the classic Master Theorem and the Akra–Bazzi Theorem provide a 'cookbook method' to determine their asymptotic growth relatively easily. In addition to the formalisation of the theorem itself, some automation infrastructure is provided in order to facilitate applying the theorem to concrete examples.

Linear Recurrences [40]. In contrast to divide-and-conquer recurrences, linear recurrences (with constant coefficients) are a class of recurrence relations for which the exact solutions have a very simple form and can be determined fairly easily. This work describes the (probably) first formalisation of the theory of these recurrences (both homogeneous and inhomogeneous) and an executable solver for them. However, the exact solutions can contain irrational algebraic numbers, which are difficult to handle. For the use case where one is only interested in asymptotic upper bounds of the solution, I therefore also implemented a certifying approach that uses ideas from analytic combinatorics and tools from complex analysis to prove 'Big-O' bounds. The exact solver relies on a formalisation of executable algebraic numbers by Thiemann et al. [94] and the certifier relies on a formalisation of executable complex-root counting by Li et al. [73]

Analytic Number Theory [29]. This is by far the largest individual formalisation work presented in this thesis: a formalisation of most chapters of Apostol's classic textbook *Introduction to Analytic Number Theory*. Analytic number theory arose in the mid-19th century in the study of the distribution of prime numbers, and this is still the application that it is best known for. Its most famous theorems are the *Prime Number Theorem* (on the asymptotic distribution of primes) and *Dirichlet's Theorem* (on the infinitude of primes in arithmetic progressions), and its most important tools are Dirichlet series and the Riemann ζ function. All of this (and much more) is covered in Apostol's book and was formalised by me as part of this work.

Related Contributions Without Formal Publications. In addition to the four articles that were just mentioned, I also have other contributions to the Isabelle/HOL distribution and the *Archive of Formal Proofs* that are *not* part of any formal publication. I would like to briefly mention some that are related to this thesis because they are either directly used by the other work described in it or also concern the subject of asymptotics.

In the Isabelle distribution, there are:

- the Γ function
- the radius of convergence of a power series and various summability tests
- the generalised binomial theorem
- the connection between formal power series and complex functions

The last item here is particularly significant since a similar connection between a class of formal series and analytic functions will also appear in my work on analytic number theory in Chapter 8.

As for the *Archive of Formal Proofs*, I specifically have entries on:

- Stirling's formula (the asymptotics of $n!$ and Γ) [31]
- Catalan numbers, including an analytic-combinatorics-inspired derivation of their asymptotics [26]
- basic properties of some special functions like the error function and the Lambert W function, including their asymptotic behaviour [32, 36]
- the Euler–MacLaurin formula (relating sums to integrals) [33]
- advanced properties of Bernoulli numbers (building on work by Bulwahn) [18]

The last two items again play an important role in analytic number theory, specifically in connection with the Hurwitz and Riemann ζ functions.

5 Semi-Automatic Real Asymptotics

“Divergente Rækker ere i det Hele noget Fandensskab,
og det er en Skam at man vover at grunde nogen Démonstration derpaa.
Man kan faae frem hvad man vil naar man bruger dem,
og det er dem som har gjort saa megen Ulykke og saa mange Paradoxer.

Divergent series are, all in all, an abomination,
and it is shameful that one should dare base any proof upon them.
Using them, one can obtain whatever one wishes
and they have created so much misfortune and so many paradoxes.”

— Niels Henrik Abel, in a letter to Bernt Holmboe (1826)

“The series is divergent;
therefore we may be able to do something with it.”

— attributed to Oliver Heaviside

It is not surprising that when formalising mathematics with a proof assistant, one frequently has to prove limits and other asymptotic properties of concrete functions. On the simple end of the spectrum, there are e.g.

$$\lim_{n\to\infty}(n+c) \qquad \lim_{n\to\infty}\frac{n}{n^2+1} \qquad c\,x^n \in O(e^x)\,,$$

which a human mathematician would regard as completely trivial. On the other hand, there are problems such as

$$\frac{\log^n k}{k} - \frac{1}{n+1}\left(\log^{n+1}(k+1) - \log^{n+1} k\right) \;\sim\; \frac{1}{2}k^{-2}\log^n k \qquad \text{for } k \longrightarrow \infty$$

$$\lim_{x\to\infty}\left(1-\frac{1}{b\log^{1+\varepsilon} x}\right)^p \left(1+\frac{1}{\log^{\varepsilon/2}\left(bx + x\log^{-1-\varepsilon} x\right)}\right) - \left(1+\frac{1}{\log^{\varepsilon/2} x}\right) \;=\; ?$$

which are fairly tedious to do even on paper.

Recall that $\log^k x$ means $(\log x)^k$, *not* the iterated logarithm.

Unfortunately, in my personal experience, *all* of these are tedious to do in a theorem prover. The ones that are easy on paper tend to be easier in the theorem prover as well, but still: even for one of the trivial problems listed above, one might need a few lines of formal proofs and remember the names of all the required library theorems. Since such simple problems tend to arise very often, this effort accumulates and becomes quite cumbersome in any serious mathematical formalisation project involving asymptotic arguments. As for the more difficult problems, my first version of the formal proof of the last example above required 700 lines of formal proofs and more than a week of time (see the attached article in Chapter B in the appendix).

On the other hand, unverified symbolic computer algebra systems such as Mathematica [102] or Maple [97] have been solving such problems fairly well since the 1990s. The painful experience of having to prove asymptotic statements like the ones above over and over again caused me to look into *how* modern computer algebra systems solve such problems. This brought me to the work by Gruntz [50], which ultimately led me to the *Multiseries expansions* by Shackell et al. [87].

The final outcome of my work in this area was then a proof-constructing procedure[1] that can compute Multiseries expansions for a wide variety of real-valued functions, from which the desired asymptotic properties of the function (such as limits or 'Big-O' behaviour) can then be read off. I called the procedure *semi-automatic* because it relies on Isabelle's existing automation to solve the question of whether a given expression is or is not equal to zero, and if it is not, whether it is positive or negative. When this automation fails, the user must step in and supply additional facts until the automation can deduce this information. In my experience, this does not happen particularly often in practice.

This procedure has since become invaluable to me in automating both the abovementioned multitude of trivial asymptotic problems and the more difficult problems that arise occasionally. In particular, it facilitates working with integrals or infinite sums: rigorous formal proofs require establishing integrability or summability explicitly, and this can often be done most easily by performing a comparison test with a suitable 'Big-O' bound.

For example, the first of the last two examples given above arises in proving the well-definedness of the Stieltjes constants (the Laurent series coefficients of $\zeta(s)$ at $s \longrightarrow 1$). These are defined as an infinite sum whose convergence is most easily shown by noting that the summand is $O(k^{-1.5})$, and $k^{-1.5}$ is clearly summable. This 'Big-O' estimate involves cancellation, so it would have been quite tedious to prove by hand in Isabelle, whereas the automation can now do it easily.

The other of the two examples arose in the formalisation of the *Akra–Bazzi theorem* (which will be covered in Chapter 6). As mentioned above, the first, manual proof for this limit was over 700 lines long and took over a week to write (not counting quite some time before that to understand how to even prove it rigorously on paper). This experience was in fact my original motivation for creating `real_asymp` – which can now prove it fully automatically in less than a second.

[1] 'Proof-constructing' means that the procedure builds an actual proof by (at the end of the day) interacting with the Isabelle kernel, as opposed to acting as or using some kind of oracle (such as computational reflection). This means that it can also construct a proof object, although this is usually not enabled in Isabelle. In principle, this does however open up the possibility of replaying such proofs in other theorem provers.

5.1 Multiseries

The basic tool upon which `real_asymp` is based is the concept of *Multiseries.* A Multiseries is a formal power series in n variables, with real coefficients and exponents. The variables themselves are functions $b_1(x), \ldots, b_n(x)$ of a real variable x. Consequently, the monomials of a Multiseries are of the form $c\, b_1(x)^{e_1} \ldots b_n(x)^{e_n}$. The vector of these n functions is called an *asymptotic basis.*

By convention, we always consider series at the point $x \longrightarrow \infty$. The functions in an asymptotic basis are also, by convention, required to tend to ∞ as $x \longrightarrow \infty$. Furthermore, they are sorted descendingly by growth in the sense that $\log b_{i+1}(x) \in o(\log b_i(x))$. This implies, in particular, that $b_i(x)^\varepsilon$ grows faster than $b_{i+1}(x)^e$ for any $\varepsilon > 0$ and any e. Due to this, asymptotic comparisons of monomials are very easy: comparing the growth of two monomials is equivalent to comparing their exponent vectors lexicographically.

The following sections will give a brief informal explanation of Multiseries with the aim to convey the general picture. More detailed information on these theoretical aspects can be found in the works of Richardson et al. [87], Shackell [90], and Van der Hoeven [62].

Formal Definition

Formally, Multiseries can be seen simply as a function from exponent vectors to coefficients, i.e. $\mathbb{R}^n \longrightarrow \mathbb{R}$. However, additional restrictions on the support of this function must be made. In our setting, we will always deal with Multiseries with finitely-generated support, i.e. whose support is contained in a set of the form

$$\{\alpha + \lambda_1\beta_1 + \ldots + \lambda_k\beta_k \mid \lambda_i \in \mathbb{N}\} \quad \text{for fixed vectors } \alpha \in \mathbb{R}^n \text{ and } \beta_1, \ldots, \beta_k \in \mathbb{R}^n_{\leq 0}.$$

This formal view is, however, not particularly useful for implementation purposes. Since the end goal is not to have a nice algebraic formalisation of Multiseries but rather to have a practical tool for asymptotic reasoning, the Isabelle formalisation does not define Multiseries this way. Rather, it follows the same approach that is used in the Maple implementation by Salvy [97], which uses a more application-driven hierarchical definition. I will now explain what this representation looks like.

Representation

Let us first consider ‘normal’ generalised power series – that is, power series in one variable x where exponents can be any real number. These are essentially Multiseries with only one basis element, which is x. Under reasonable support conditions, they can be thought of and represented as a linear sequence of coefficients and exponents, where the exponents are required to decrease strictly. For instance, the power series expansion of $e^{1/x}$ at $x \longrightarrow \infty$ can be represented as the sequence $1 + x^{-1} + \frac{1}{2}x^{-2} + \ldots$

In a functional setting, the obvious representation of such a series is simply a (possibly infinite) list of pairs of real numbers (coefficient and exponent). In Isabelle/HOL, that type would be written as $(\mathit{real} \times \mathit{real})\ \mathit{llist}$ (since possibly-infinite lists are called ‘lazy lists’ in functional programming).

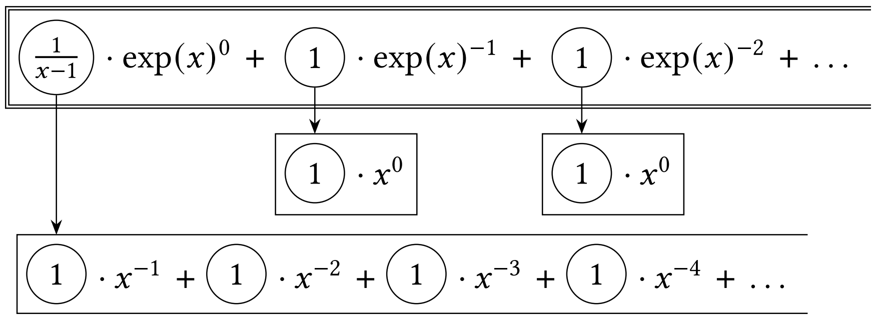


Figure 5.1: A hierarchical illustration of the Multiseries of the function $\frac{1}{e^x-1} + \frac{1}{x-1}$ for $x \longrightarrow \infty$ w.r.t. the basis $b_1(x) = \exp(x)$ and $b_2(x) = x$.
The uppermost layer (represented by a double rectangle) is a power series in b_1. Its coefficients are again Multiseries w.r.t. the singleton basis b_2, which are represented by the other rectangles below. The circled digits '1' in the in the single-framed rectangles are then simply Multiseries w.r.t. the empty basis (i.e. constants).

For Multiseries, this simple linear representation is usually not possible. Consider the following example:

$$\frac{1}{x-1} + \frac{1}{e^x-1} \quad \sim \quad \sum_{i=1}^{\infty} x^{-i} + \sum_{i=1}^{\infty} \exp(x)^{-i}$$

The function on the left-hand side has the Multiseries expansion on the right-hand side at $x \longrightarrow \infty$. This series is a infinite sequence of monomials of the form x^{-i}, followed by an infinite sequence of monomials of the form $\exp(x)^{-i}$. If we implemented the series as simply a lazy list of monomials, the information on the second term would be completely lost: if we were to subtract $\frac{1}{x-1}$ from our series, we should be able to find that the leading term is now $\exp(x)^{-1}$ – but with the linear representation, we would be left with nothing but an infinite number of zero terms.

In general, it is even possible to have e.g. an infinite sequence of infinite sequences of monomials. To accommodate this more intricate structure, one can adopt a hierarchical view:

- A Multiseries with an empty basis is just a single real number.
- A Multiseries with a basis $b_1, \ldots, b_n$ is a generalised power series in $b_1(x)$ whose coefficients are Multiseries with the basis $b_2, \ldots, b_n$.

The following pseudocode illustrates roughly how Multiseries are defined in Isabelle/HOL:

$$\begin{aligned} [\,]\ \text{multiseries} &= \text{real} \\ [b_1, \ldots, b_n]\ \text{multiseries} &= ([b_2, \ldots, b_n]\ \text{multiseries} \times \text{real})\ \text{llist} \end{aligned}$$

Figure 5.1 illustrates this principle on our earlier example. The list of the infinitely many monomials x^{-i} together forms the first monomial of the overall Multiseries, and each of the monomials $\exp(x)^{-i}$ is another monomial after that. Thus, all the asymptotic information is preserved: if we were to subtract $\frac{1}{x-1}$ from this series, the first monomial of the Multiseries would disappear and the new leading term would be $\exp(x)^{-1}$, as it should be.

Connecting Series and Functions

Although Multiseries are convergent in many cases, they *can* be divergent[2] (cf. e.g. Stirling's formula for the Γ function). This means that looking at a particular Multiseries, it is not always immediately obvious which actual real function (if any) it corresponds to.

However, in our case, we are only interested in attaching a Multiseries to a known function, which can be done in analogy to Poincaré series. For this purpose, let us first again consider the easier setting of 'normal' power series where we only have one basis element x. For a function $f : \mathbb{R} \to \mathbb{R}$ and such a series,

$$f(x) \sim \sum_{i=0}^{\infty} c_i x^{e_i}$$

holds in the sense of a Poincaré expansion iff

$$\left(f(x) - \sum_{i=0}^{N} c_i x^{e_i}\right) \in o\big(x^{e_i}\big) \qquad \text{for all } N \in \mathbb{N}\ .$$

A nice alternative coalgebraic formulation of this is the following inference rule:

$$\frac{f(x) \in O\big(x^{e_0}\big) \qquad f(x) - c_0 x^{e_0} \ \sim\ \sum_{i=0}^{\infty} c_{i+1} x^{e_{i+1}}}{f(x) \ \sim\ \sum_{i=0}^{\infty} c_i x^{e_i}}$$

If we interpret this inference rule as the definition of a coinductive relation $\sim$, we obtain exactly the same predicate as with the Poincaré definition, but can use corecursion to reason about it. For instance, the correctness lemmas for operations on series become straightforward coinduction proofs.

A very similar inference rule can be used to define a $\sim$ relation on Multiseries. The main difference is that since the coefficients are then again Multiseries (with a smaller basis), these also need to recursively satisfy the $\sim$ relation for the functions that they represent.

The Anatomy of the Basis Functions

Let us now turn to what form the asymptotic bases have concretely. In our setting, every basis always contains the function x. After this x, we can have a list of increasingly iterated logarithms of x. Before the x, we can have exponentials of functions that already have an expansion w.r.t. the smaller basis elements after it. For the purpose of illustration, the following is a valid asymptotic basis:

$$\begin{gathered}\exp(\exp(x^2) + \exp(\sqrt{x})),\ \ \exp(x^2 - x),\ \ \exp(x),\ \ \exp(\sqrt{x}),\\ x,\ \ \log x,\ \ \log\log x,\ \ \log\log\log x\end{gathered}$$

[2]In case the reader now feels some unease concerning the earlier quotation by Abel, they need not worry. Abel was referring to unsound manipulation of divergent series. No such issues arise in our context, and even if they did, the theorem prover would force us to handle them in a sound way.

This approach is sufficient to treat a wide variety of functions, including everything built from

- basic arithmetic
- exponentials, logarithms, roots
- the 'absolute value' and sgn functions
- the trigonometric functions sin, cos, tan as long as their argument does not tend to ∞
- arctan, Γ (the Gamma function), ψ (the Polygamma functions), erf (the error function), Bessel functions, the Riemann ζ function[3]

This can also be extended to other classes of functions (such as implicit functions, inverse functions, and solutions of differential equations [90]) but that is certainly beyond the scope of the present work.

5.2 Implementation

Shackell et al. [87] sketch a bottom-up syntax-directed procedure to produce Multiseries expansions for a real-valued function that is given as an explicit expression. The basis of the Multiseries is computed on-the-fly, starting with the basis x and adding new basis elements as needed.

The Isabelle implementation of `real_asymp` follows exactly this approach. Of course, the traditional implementation of the algorithm by Shackell et al. is in the context of a computer algebra system, where only a *result* has to be produced. Since we are working in a theorem prover, we must produce *theorems* in every step. `real_asymp` must prove the well-formedness of all the asymptotic bases and Multiseries it produces, and that the Multiseries is indeed an expansion for the given function. To do this, an Isabelle/HOL function was defined and proven correct for each of the operations in the syntax-directed procedure (under some suitable preconditions). The actual `real_asymp` procedure then only has to

- construct the Multiseries for the function expression bottom-up,
- plug the correctness results of the relevant operations together,
- prove the arising preconditions (which are often of the form 'this real-number term is non-zero' or 'this real-number term is greater than 1'),
- evaluate the resulting Multiseries as far as needed (usually until the leading term can be read off),
- interpret the result (e.g. if the leading term is of the form $c(x)b_1(x)^e$ with $e < 0$, we know that the function tends to 0).

This approach naturally leads to a *proof-producing* procedure: every step of the reasoning done by `real_asymp` passes through the Isabelle kernel.

The evaluation of the Multiseries requires a way to lazily expand equational definitions until a certain point, which is why `real_asymp` contains a simple ad-hoc proof-producing engine for lazy evaluation of Isabelle/HOL terms. Since the evaluation must produce Isabelle

[3]Of these special functions, only arctan, Γ, and erf were implemented in `real_asymp`, and only partially.

theorems, there is significant performance overhead in this step compared to e.g. a computer algebra system. Sharing of common subexpressions is currently also not supported, since the combination of sharing and producing theorems is not straightforward.

One problematic aspect of the entire implementation is the aforementioned determination of signs and zeroness. Even for constants built from a fairly restricted set of operations, determining zeroness is not known to be decidable. For the most basic setting of exp–log functions, Richardson [86] gave a partially correct algorithm whose termination is contingent on *Schanuel's conjecture*, a deep conjecture in transcendental number theory. In the considerably more liberal setting that we have, zero equivalence (and thus also sign determination) are known to be undecidable due to another theorem by Richardson [85]. Due to the complexity of the problem, the `real_asymp` procedure simply uses Isabelle's simplifier and optionally Isabelle's `approximation` package [63] as heuristics to solve such problems, and fails if it is unable to determine the signs that way. This is in line with what computer algebra tools do: Maple, for instance, uses a number of heuristic and probabilistic methods to determine if an expression is zero but does not implement anything along the lines of Richardson's algorithm.[4]

5.3 Asymptotic Interval Arithmetic

Oscillating functions such as $\sin(x)$ or $\lfloor x \rfloor$ for $x \longrightarrow \infty$ do not have Multiseries expansions so that the approach as presented above does not work for them. Generalising the Multiseries framework to encompass such functions is difficult; some approaches exist (e.g. *measured limits* by Salvy and Shackell [89]) but are probably too complicated to justify the additional complexity for an application such as ours. I therefore implemented another fairly straightforward approach that builds on top of Multiseries in a modular way: when `real_asymp` encounters a possibly oscillating expression (such as $\sin(f(x))$ or $\lfloor f(x) \rfloor$ where $f(x)$ does not tend to a finite value), it uses reasonable asymptotic lower and upper bounds (e.g. -1 and 1 resp. $f(x) - 1$ and $f(x)$ in the previous two examples). This requires implementing a kind of interval-arithmetic approach for asymptotic expansions, i.e. all the different cases in the syntax-directed procedure mentioned previously now need to be modified to handle the cases where the input is not just a function with a single Multiseries expansion but possibly one with an asymptotic lower and upper bound, each having a separate Multiseries expansion. This adds a considerable amount of implementation effort.

This approach is certainly not *complete* since the usual problem of interval arithmetic occurs: wrapping. For a drastic example, consider the function $\sin(x) - \sin(x)$. Since `real_asymp` does not simplify the input term, it infers the bounds $[-1, 1]$ for each of the $\sin(x)$ terms, leading to an overall result of $[-2, 2]$ instead of just 0. A less obvious example would be $\sin(x) + \cos(x)$ with the inferred bounds $[-2, 2]$ while the best possible bounds are $[-\sqrt{2}, \sqrt{2}]$. However, this approach can still handle many practical examples – in particular, it is very useful in gauging the asymptotic error made by rounding, such as $\sqrt{\lfloor x \rfloor} = \sqrt{x} + O\big(\frac{1}{\sqrt{x}}\big)$.

[4]according to an answer on Math StackExchange by Jacques Carette, a former Maple developer (`https://math.stackexchange.com/questions/3606561`)

5.4 Related Literature

There is little other work on automating proofs for asymptotic statements in a theorem prover. Avigad and Donnelly [6] developed a decision procedure for the fragment of linear Big-O expressions (which has unfortunately never been integrated into a theorem prover).

In my publication on the Akra–Bazzi theorem [30], I developed some simple, more heuristic automation for Landau Symbols (see Chapter 6 for more details on this).

Publications on formalisation techniques for asymptotics are also relatively rare. Hölzl et al. [64] describe how Isabelle/HOL uses *filters* to denote and reason about limits and other asymptotic properties (which is also heavily used in my work). Harrison [59] describes the use of *nets* for the same purpose in HOL Light. Affeldt et al. [1] describe a formalisation approach for 'Big-O'-style asymptotics in Coq, including some tactics to facilitate making assumptions such as 'x is big enough' in an idiomatic way.

Outside the realm of theorem proving, the most relevant literature that I am aware of are the aforementioned article by Shackell et al. [87] and dissertation by Gruntz [50], and, for a more comprehensive view on the matter, a book by Shackell [90]. There is also a book by Van der Hoeven [62], which takes the somewhat different route of *Transseries*, but I found the Multiseries approach from the previously mentioned sources more accessible for my purposes.

5.5 Future Work and Outlook

There are still some minor issues with the current version of `real_asymp` that should eventually be remedied:

- Sometimes, the expressions arising in intermediate steps become so complicated that the simplifier is no longer able to recognise them as zero. It is not clear if and how this can be avoided.
- While there is a limited plugin system to add support for new special functions at the 'user level', doing so is currently very tedious and only done for the Γ function as a proof of concept.
- The functions exp, log, sin, etc. all satisfy nice equations w.r.t. addition and/or multiplication, which makes deriving Multiseries expansions for them fairly easy. For functions such as Γ and arctan, the situation is more complicated and the support for these is therefore still incomplete. Shackell [90] briefly explains how to handle this case, but without giving a proof. I was not able to find an actual proof of this anywhere else in the literature either. Moreover, it would unfortunately be difficult to integrate that approach with the current setting of `real_asymp` on a purely technical level as well.

When I spent more time formalising mathematics after the development of `real_asymp`, I also came across the following more 'high-level' ideas for future work:

- In exactly the same way as `real_asymp`, one could write a procedure to determine Laurent series expansions for meromorphic complex functions. In fact, the implementation would actually be significantly more straightforward since Laurent series are conceptually much simpler than Multiseries seeing as there is no 'asymptotic basis' –

just integral powers of x. Such a method could automate some of the tedious tasks that arise when doing complex analysis in Isabelle, such as limits, pole cancellation, and determination of residues.

- It would be desirable for `real_asymp` to also work with partial asymptotic information, e.g. the user could prove that $f(x) = x + O(1)$ and then ask for the limit of $e^{f(x)}$. This would, however, require a full rewrite of much of the existing code, and it is not quite clear how this would interact with the existing 'asymptotic interval arithmetic'.

The first item in particular seems very promising, since it would probably be comparatively little work and result in a very useful tool that could, among other things, simplify some of the complex-asymptotic reasoning in my work on the formalisation of analytic number theory.

Looking at other systems than Isabelle/HOL, it should be noted that the mathematical prerequisites of `real_asymp` are very moderate: limits (of course), convergent power series, Landau symbols, and ideally a framework for coalgebraic definitions and reasoning. The formalisation behind `real_asymp` itself is also relatively small and consists entirely of simple and straightforward proofs. It should therefore be possible to implement a similar procedure in any system with a library of classical analysis (such as Coq, Lean, or HOL Light) with little formalisation effort. However, the *implementation* effort was very substantial in Isabelle and may well be so in other systems as well.

6 Divide-and-Conquer Recurrences

“Entzwey’ und gebiete! Tüchtig Wort:
Verein’ und leite! Beßrer Hort.

Divide and Conquer! Fine words.
Unite and lead! A better tenet.”

— Johann Wolfgang von Goethe (1814)

The famous *Master Theorem* is part of any undergraduate lecture on algorithms. It can be used for the running time analysis of a large class of classic divide-and-conquer algorithms, which have the following form:

Input: A problem of size n
Algorithm: For n sufficiently small, solve non-recursively. Otherwise:

1. split the problem into m subproblems of size n/k
2. solve each problem recursively
3. combine the m subsolutions into a solution for the original problem

If we denote the time required by this algorithm with $T(n)$ and the time required to recombine the solutions in the last step of each recursion with $g(n)$, this gives us the recurrence

$$T(n) = m \cdot T(n/k) + g(n)$$

for sufficiently large n. The Master Theorem allows us to determine the asymptotic growth of $T(n)$ in this setting under some mild conditions. The upper half of Table 6.1 lists some notable example applications of the Master Theorem.

The theorem was introduced by Bentley et al. [10] in 1980. Cormen et al. popularised it under the name *Master Theorem* in their famous textbook. For reasons that will be discussed in the next section, I instead formalised a generalisation by Akra and Bazzi [2] from 1998 that applies to more general recurrences (such as the ones in the lower half of Table 6.1).

6.1 The Formalised Theorem

My original plan was to formalise the *Master Theorem*, but the proofs found in textbooks seemed less than ideal for formalisation. The Master Theorem is divided into three different cases that are proven completely separately. Additionally, due to the discrete nature of the inputs, algorithms cannot always evenly divide subproblems in practice – for instance, the Merge Sort recurrence is not *really* $T(n) = 2T(n/2) + O(n)$, but rather $T(n) = T(\lfloor n/2 \rfloor) +$

Algorithm	Recurrence	Asymptotics
Binary search	$T(n) = T(n/2) + O(1)$	$\Theta(\log n)$
Merge Sort	$T(n) = 2T(n/2) + O(n)$	$\Theta(n \log n)$
Karatsuba multiplication	$T(n) = 3T(n/2) + O(n)$	$\Theta(n^{\log_2 3})$
Ham-Sandwich trees	$T(n) = T(n/2) + T(n/4) + 1$	$\Theta(n^{\log_2(1+\sqrt{5})-1})$
Median-of-medians	$T(n) = T(2/10 \cdot n) + T(7/10 \cdot n + 6) + O(n)$	$\Theta(n)$

Table 6.1: Three divide-and-conquer recurrences related to algorithm analysis with their asymptotics as given by the Master Theorem, and two for which the Master Theorem does not apply (but the Akra–Bazzi Theorem does).

$T(\lceil n/2 \rceil) + O(n)$. This rounding introduces additional difficulties and makes it impossible to find an exact closed-form solution in most cases (even though it does not change the asymptotic result at all). In informal presentations on paper, this aspect is often either ignored completely or handled through additional case distinctions, which blows up the proof size even more.

From my own formalisation experience, I felt that a more unified and high-level approach would be easier to formalise, and it was this that led me to the Akra–Bazzi theorem. This theorem provides one single unified solution (albeit a more complex one) with no case distinctions, and it solves the issue of rounding by allowing the recursive calls to have arbitrary 'error terms' as long as they are small enough. This allows not only rounding up and down, but also adding arbitrary constants, or even terms such as $\lfloor n/2 + \sqrt{n} \rfloor$. It also generalises the Master Theorem in another way, namely that the subproblems are not required to all have the same size.

My formalisation follows the presentation by Leighton [72], which generalises the result by Akra and Bazzi even further and also provides a more elementary proof that seemed more suitable for formalisation since the mathematical prerequisites for it are minimal.

The precise class of recurrences that the Akra–Bazzi theorem, as presented by Leighton, applies to is the following:

$$T(n) = g(n) + \sum_{i=1}^{m} a_i \cdot T(b_i n + h_i(n)) \tag{6.1}$$

with $a_i \geq 0$ and $b_i \in (0, 1)$. Additionally, the recombination cost function $g(n)$ and the error terms $h_i(n)$ need to satisfy some mild growth conditions. The resulting growth estimate is

$$T(n) \in \Theta\left(x^p \left(1 + \int_{x_0}^{x} \frac{g(u)}{x^{p+1}} \mathrm{d}u\right)\right)$$

where p is the unique real solution of $\sum a_i b_i^p = 1$. This rather unwieldy result can be simplified significantly for more concrete forms of $g(n)$ that often arise in practice, which then leads to a number of simple corollaries that have a form that is very similar to the standard Master Theorem, but more general (see Table 6.2).

Case name	Assumptions			Conclusion
Case 1 (O)	$g(n) \in O(n^q)$	$q < p$		$T(n) \in O(n^p)$
Case 1	$g(n) \in O(n^q)$	$q < p$	$f(n) > 0$ (for suff. large n)	$T(n) \in \Theta(n^p)$
Case 2.1	$g(n) \in \Theta(n^p \log^q n)$	$q < -1$		$T(n) \in \Theta(n^p)$
Case 2.2	$g(n) \in \Theta(n^p / \log n)$			$T(n) \in \Theta(n^p \log\log n)$
Case 2.3	$g(n) \in \Theta(n^p \log^q n)$	$q > -1$		$T(n) \in \Theta(n^p \log^{q+1} n)$
Case 3	$g(n) \in \Theta(n^q)$	$q > p$		$T(n) \in \Theta(n^q)$

Table 6.2: The five cases of the generalised Master Theorem as formalised in Isabelle/HOL. The setting is the same as that described in Equation (6.1), with p the unique real solution to $\sum a_i b_i^p = 1$. Which of the cases applies (if any) depends on the asymptotics of $g(n)$.

The formal proof consists of:

- proving the growth estimates for continuous recurrences of the type $\mathbb{R} \to \mathbb{R}$ (with some additional technical assumptions)
- lifting the estimates to discrete recurrences of the type $\mathbb{N} \to \mathbb{R}$ (while discharging the aforementioned assumptions)
- deriving various corollaries, such as the original Master Theorem

The advantage of this approach is that all the hard work is done in the first two steps and no case distinctions are necessary until the very last step, which is a very easy one.

The only major difficulty in the formalisation were a number of asymptotic inequalities that needed to be proven (these were briefly mentioned in Chapter 5). Leighton himself did not give a proof for them but only remarked in a footnote that they can easily be shown by Taylor expansion. My first formal proof showed the inadequacy of the existing machinery for asymptotic reasoning in Isabelle/HOL and led to the development of the `real_asymp` method mentioned in Chapter 5, which completely automates the proof of these inequalities now.

6.2 Automation

For practical applications, the generalised Master Theorem as shown in Table 6.2 is much more useful than the Akra–Bazzi theorem itself. However, in practice, it is still rather tedious to instantiate the formalised Master Theorem with all the correct values manually. I therefore also developed some automation to analyse a given recurrence and, as far as possible, instantiate the appropriate case of the Master Theorem automatically and solve all obvious proof obligations arising from it.

Many of the remaining non-obvious proof obligations are growth conditions such as $n \log n \in O(n^2)$, which are obvious to a human but beyond the capabilities of Isabelle's general-purpose automation. The `real_asymp` method from the previous chapter did not yet exist back then either. In order to tackle these proof obligations, I developed some rather basic automation for Landau symbols in the form of *simprocs*, which are plugins for Isabelle's simplifier and are therefore integrated with Isabelle's general-purpose automation – as opposed to the `real_asymp` method mentioned in Chapter 5, which must be invoked manually. In particular, there are simprocs to:

- cancel dominated terms from Landau symbols, e.g. rewriting $O(x^2 + x)$ to $O(x^2)$
- cancel common factors, e.g. rewriting $f(x)h(x) \in O(g(x)h(x))$ to $f(x) \in O(g(x))$ (requiring a heuristic to prove that $h(x)$ is non-zero for sufficiently large x)
- simplify Landau symbols involving products of powers of iterated logarithms, e.g. rewriting $x^a \log^b x \in O(x^c \log\log^d x)$ to an equivalent statement involving only the exponents a, b, c, d

Lastly, I also developed the `akra_bazzi_termination` tactic, which solves a rather technical issue: when defining a divide-and-conquer recurrence in Isabelle/HOL with the `function` package, one needs to prove termination. The `function` package can often prove termination automatically by finding a pattern in which the arguments of the recursive call get 'smaller', but this usually fails for our more complicated divide-and-conquer recursions involving division and rounding. To solve this, the `akra_bazzi_termination` tactic proves the termination of such recursion patterns automatically, which makes defining such divide-and-conquer recurrences and algorithms more convenient in Isabelle.

I used the combination of all of this machinery to give mostly automatic analyses of the asymptotic growth of the recurrences of several algorithms in Isabelle/HOL, including all the ones mentioned above. The proofs are very short and simple – they consist of a single invocation of my `master_theorem` tactic with some appropriate parameters followed by proofs of some simple side conditions, which can mostly be done by Isabelle's automation.[1]

I believe that my automation makes using 'my' Master Theorem much easier for other Isabelle/HOL users who are not as familiar with my formalisation. To support this claim, note that the author of an AFP submission has since used this work in his formalisation of a closest-pair-of-points algorithm [82], without requiring any assistance from me. Zhan and Haslbeck [103] also incorporated my machinery into their work on verifying asymptotic time complexity of imperative algorithms and told me in personal communication that they found it very convenient to use.

6.3 Related Literature

The classic textbook version of the Master Theorem and a sketch of its proof can be found e.g. in the well-known textbook by Cormen et al. [25] The Akra–Bazzi Theorem was first proven (in a weaker form than what I formalised) by Akra and Bazzi using rather elaborate mathematical tools [2]. My formalisation, on the other hand, is based on an unpublished note

[1]The parameters that need to be given to the tactic are which case of the Master theorem the recurrence falls under and – in some cases – the value of q (cf. Table 6.2). Nowadays, with `real_asymp` being available, it would be possible to try to infer these automatically as well, but I have not yet attempted to do so.

by Leighton [72], which gives a much more elementary proof for a stronger version. Bazzi and Mitter [9] later also proved an analogue of the Akra–Bazzi theorem for the asymptotic expected value of probabilistic recurrences.

In the area of theorem proving, there is no other formalisation of the Akra–Bazzi theorem or the Master theorem that I am aware of. However, based on work by Karp [67], Tassarotti and Harper [92] formalised a cookbook method for tail-bounds for probabilistic recurrences that arise in the analysis of certain randomised programs.

7 Linear Recurrences

“ Quīdam posuit ūnum pār cunīculōrum in quōdam locō […]
ut scīret quot ex eō paria germinārentur in ūnō annō
cum nātūra eōrum sit per singulum mēnsem aliud pār germināre
et in secundō mēnse ab eōrum nātīvitāte germinant.

Someone put one pair of rabbits in a certain place […]
so that he may know how many are born from that pair in a year
when it is their nature to bear another pair in a single month
and these also bear in the second month from their birth. ”

— Leonardo ‘Fibonacci’ of Pisa, *Liber abbaci* (1202)

Linear recurrences with constant coefficients[1] are a class of recurrence equations for which a closed-form solution can be found algorithmically in a comparatively easy way. The most famous example of such a recurrence are the *Fibonacci numbers* hinted at in the above quotation. In modern texts, they are usually defined by the recurrence $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_{n+1} + F_n$ and, perhaps surprisingly, they have the closed-form solution

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) .$$

Such recurrences arise in many other settings, including combinatorics and the analysis of algorithms and data structures. In particular:

- The numerators and denominators of the n-th convergent of the continued fraction expansion of the golden ratio $\varphi = \frac{1}{2}(1+\sqrt{5})$ are F_{n+1} and F_n. This is also the ‘worst case’ in the sense that they converge more slowly than any other continued fraction.
- Fibonacci numbers also appear in the performance analysis of AVL trees, and inspired the data structure of *Fibonacci heaps*.
- Similarly, the worst-case behaviour of height-balanced trees satisfies a linear recurrence [74].
- The numbers of possible syllable patterns in metres of Sanskrit prosody are precisely the Fibonacci numbers (and they were first studied in ancient India due to this). [91]
- More generally, given a regular language A, the number $|A|_n$ of words $w \in A$ of length n satisfies a linear recurrence in n.

[1]From now on, I will write ‘linear recurrence’ and will always implicitly mean ‘with constant coefficients’.

- The *complexity descriptors* in the automatic algorithm analyser $\Lambda\Upsilon\Omega$ ('LUO', or 'Lambda–Upsilon–Omega') by Flajolet et al. [43] typically satisfy linear recurrences.

I formalised the algebraic theory of linear recurrences with constant coefficients based on rational generating functions. In particular, I formalised the standard result that any such recurrence has a rational generating function, from which the closed-form solution can then be read off. The entire process – starting with a recurrence and ending with a closed-form solution – was formalised in such a way that it yields an executable solver.

7.1 Definitions and Scope

A linear recurrence equation with constant coefficients $c_0, \ldots, c_n$ has the form

$$c_k a_{n+k} + \ldots + c_0 a_n = f(n) \quad \text{for any } n \in \mathbb{N}\,,$$

where $f(n)$ is called the *inhomogeneous part.* If $f(n)$ is identically zero, the recurrence is called *homogeneous.* For the most part, the properties that were formalised work in any field, but since we will need to fully factor polynomials, it is convenient to work in an algebraically closed one. The executable part (which requires an executable factorisation algorithm) was only set up for the field of complex numbers.

Every homogeneous recurrence has a polynomial–exponential closed-form solution, i.e.

$$a_n = p_1(n)\lambda_1^{-n} + \ldots + p_k(n)\lambda_k^{-n}$$

where the λ_i are roots of the *characteristic polynomial* $c_k X^k + \ldots + c_0$ of the recurrence. The p_i are polynomials in n whose degree depends on the multiplicity of the corresponding root λ_i.

One standard way to show this (and this is the route that was taken in my formalisation) is to first show that any such recurrence has a *rational generating function.* That is, if we form the generating function

$$A(z) := \sum_{n=0}^{\infty} a_n z^n$$

in the ring of formal power series, then $A(z)$ is *rational* – there exist polynomials P and Q such that $A(z) = P(z)/Q(z)$. These polynomials can in fact easily be determined from the coefficients c_i and the initial values $a_0, \ldots, a_{k-1}$. The next step is then to factor the denominator polynomial Q and perform partial fraction decomposition. After this, a closed form for the coefficients of each summand can easily be read off, and summing up all of them leads to a closed form for a_n.

For inhomogeneous recurrences, the exact same approach can be taken as long as the inhomogeneous part $f(n)$ itself has a rational generating function (which is equivalent to it being a polynomial–exponential function in the above sense).

7.2 Implementation

As mentioned before, obtaining an executable solver requires an executable algorithm for polynomial factorisation. As the Fibonacci example shows, even for simple recurrences with rational coefficients, the roots can be irrational algebraic numbers. Therefore, an executable formalisation of algebraic numbers is also required. I made use of the algebraic number formalisation by Thiemann et al. [94], which comes with a verified factorisation algorithm as well.

The executable part of my formalisation then consists of executable Isabelle/HOL functions that take some representation of a recurrence (and, in the latter case, the polynomial–exponential inhomogeneous part) and return the rational generating function (represented as the pair of numerator and denominator polynomials). This part is still independent of the formalisation by Thiemann et al. Then, there is a function that performs the factorisation and partial fraction decomposition and determines the closed form solution from the decomposition. This part makes use of the factorisation algorithm and the executable arithmetic on algebraic numbers provided by Thiemann et al.

These are then combined into functions that perform the entire process, complemented by pretty-printing functions that convert the internal representation of the closed-form solutions into a human-readable format. This was evaluated on a number of examples in the article and works without problems for all the standard examples like Fibonacci numbers. However, even for relatively low degrees of about 5, one can easily find pathological recurrences for which this code never terminates because the arithmetic on algebraic numbers is too slow since the degrees of the polynomials attached to them grow very quickly. This is partially due to the fact that the verified implementation by Thiemann et al. is, of course, not as heavily optimised as the ones in computer algebra systems like Mathematica. However, as the evaluation section in the article shows, even systems like Mathematica struggle with such pathological examples at degrees of about 9 or more.

7.3 Certifying Asymptotic Upper Bounds

As explained in the previous section, the exact closed-form solutions of linear recurrences can be unwieldy and difficult to compute due to the presence of irrational algebraic numbers and poor performance at higher degrees. However, for applications such as analysis of algorithms and data structures, it may well be enough to obtain a good asymptotic upper bound on the solution (e.g. showing that it is at most quadratic in growth). I therefore used ideas from analytic combinatorics [42] to provide an executable verified Isabelle/HOL procedure to certify such 'Big-O' bounds efficiently.

The basic idea is this: each root λ of the characteristic polynomial contributes a summand of order $n^{k-1}|\lambda|^{-n}$ to the solution, where k is the multiplicity of λ. Therefore, the overall growth of the solution is determined by the *dominant* roots – those that are closest to the origin. Let r be the distance of the dominant roots from the origin and k the maximum multiplicity among the dominant roots. Then the solution is clearly $O(n^{k-1}r^{-n})$. Therefore, if we want to prove that the solution is $O(n^K R^{-n})$, we need only show that either $R < r$, or that $R = r$ and

$k - 1 \leq K$. In other words: consider the circle of radius R around the origin. Then there are no roots strictly inside the circle, and all the roots directly on it have at most multiplicity $K + 1$. These properties can be checked easily without factoring the polynomial, using a square-free factorisation algorithm (as formalised by Thiemann et al. [95]) and a complex root-counting algorithm (formalised by Li et al. [73]). As long as the input coefficients are all rational, no arithmetic on irrational numbers is required.

Although this approach follows very naturally and obviously from the closed form of linear recurrences, it can also be regarded from the more general viewpoint of analytic combinatorics[2], which provides more context and insight: the asymptotics of a sequence of numbers can be determined by considering the dominant singularities of their generating function (when viewed as an analytic function in the complex plane). A singularity at a distance r from the origin contributes a term of order $f(n)r^{-n}$, where $f(n)$ is a subexponential term whose precise shape depends on the nature of the singularity. For meromorphic functions (including rational ones, such as here), these singularities are poles, whose contribution is very easy to determine, leading to the above method.

7.4 Related Literature and Applications

I am not aware of any other work on the general theory of linear recurrences or rational generating functions in a theorem prover. However, in their recent work on formalising Apéry's Theorem, Mahboubi et al. [75] studied some particular complicated recurrences with *polynomial* (not constant) coefficients in Coq, building on tools developed by Chyzak et al. [23].

Outside the context of proof assistants, the theory of linear recurrences and the generating-function approach to solving them can be considered folklore and is typically taught in undergraduate courses on discrete mathematics. Details on this can be found in various textbooks [48, 49]. The analytic combinatorics perspective mentioned in the last section is also explained in the book of the same name by Flajolet and Sedgewick [42].

Unverified solvers for linear recurrences are widely available (both standalone such as PURRS [8] and integrated into computer algebra systems such as Mathematica [102] and Maple [97]). These typically support a considerably larger class of recurrences than what I formalised, including systems of equations.

As for applications of my formalisation: I recently began work on a draft article on a (mostly finished) formal analysis of the worst-case height of height-balanced trees that makes use of the present linear-recurrence formalisation. My work builds on work by Luccio and Pagli [74] and includes new (and fully formalised) results on the precise asymptotics of the worst-case relationship between the size and the height of such a tree.

[2]This insight is not particularly deep, let alone novel. It is fairly obvious given some knowledge of introductory analytic combinatorics, cf. for instance the textbook by Flajolet and Sedgewick [42].

8 Analytic Number Theory

“ Einstein said ‘God does not play dice with the universe.’
I would like to think that Erdős and Kac replied:
‘Maybe so, but *something* is going on with the primes.’ ”

— Carl Pomerance on the *Erdős–Kac Theorem* (1997)

“ Die Zahlentheorie ist nützlich, weil man mit ihr promovieren kann.

Number theory is useful because one can obtain a PhD with it. ”

— Edmund Landau, *Vorlesungen über Zahlentheorie* (1927)

As an advanced case study of asymptotic arguments in Isabelle/HOL, I formalised a large amount of *analytic number theory*, following Apostol’s famous textbook [3] for the most part. As the name suggests, analytic number theory applies methods from analysis (complex analysis in particular) to number theory. The vast majority of the book focuses on multiplicative number theory, which includes topics such as divisors, square-freeness and – most importantly – prime numbers. The formalisation comprises both mathematical tools such as Dirichlet series and characters and numerous applications thereof. In total, the vast majority of Apostol’s book (except for the exercises) was formalised and in some cases the formalisation even goes beyond the scope of the book.

8.1 Formalised Results

To give only some examples of the material that was formalised:

- basic arithmetic functions such as Liouville’s λ and Möbius’ μ
- Dirichlet series (both formal and analytic and the connection between the two)
- the Riemann and Hurwitz ζ functions and the Dirichlet L functions
- multiplicative characters, especially Dirichlet characters
- the Chebychev functions ϑ and ψ and their properties
- Mertens’ Theorems
- the prime number theorem (PNT) and Dirichlet’s theorem on the infinitude of primes in arithmetic progressions
- elementary consequences of the PNT, such as the asymptotics of $\mathrm{lcm}(1, \ldots, n)$ and sharp bounds for Euler’s φ function

- the distribution of square-free and coprime numbers

Although some of these results had been formalised before (see Section 8.2), most of them had not. The express goal of this project was the formalisation of interesting advanced results, but not in isolation: the results should be built on top of a broad library of basic results that facilitates work in the formalisation of number theory in the future.

This case study also demonstrated the usability of my asymptotic notation and automation for the formalisation of 'proper' mathematics, and it resulted in several ideas for extensions and improvements to these tools.

8.2 Related Literature

There is a decent amount of prior formalisation work on number theory, both in Isabelle and in other systems:

- Basic facts such as the concept of a prime and the fundamental theorem of arithmetic are available in most proof assistants, and have been for a long time. One of the earliest (perhaps even *the* earliest) was the one by Boyer and Moore in NQTHM published in 1979 [15].
- The law of quadratic reciprocity is also available in all major systems, the first one probably being the formalisation in NQTHM due to Russinoff [88].
- Bertrand's postulate was formalised by a great number of people, including Harrison in HOL Light, Théry in Coq [93], Riccardi in Mizar [84], Carneiro in Metamath [19], Asperti and Ricciotti in Matita [5], and Biendarra and myself in Isabelle/HOL [11].
- The elementary Selberg–Erdős proof of the PNT was formalised in Isabelle/HOL by Avigad et al. [7] and by Carneiro in Metamath [20].
- The complex-analytic proof of the PNT was formalised by Harrison in HOL Light [58], along with some more basic lemmas.
- Dirichlet's Theorem on primes in arithmetic progressions was formalised both by Harrison in HOL Light [57] and by Carneiro in Metamath [20].

The proof by Avigad et al. in 2004 was the first formal proof of the PNT and represents a major milestone in the theorem proving community. Their effort was a *tour de force* of formalisation in Isabelle/HOL at a time when the library and tooling of the system was much less mature than it is now. Unfortunately, their proof development consists of almost 27,000 lines of unstructured tactic scripts and was never submitted to the *Archive of Formal Proofs*. The development has therefore since fallen into disrepair. According to personal communication with Avigad, it is probably not worth the maintenance effort to make it work with current versions of Isabelle again. Hence, it is now mainly of historic interest. However, it must be noted that a considerable number of the lemmas upon which their formalisation was built has been incorporated into the Isabelle distribution, most notably its number theory library.

Harrison's analytic proof of the PNT was then another milestone since no other system had ever come close to having a library of analysis (especially complex analysis) capable of supporting such a development. Even nowadays, eleven years later, only Isabelle/HOL can rival HOL Light's complex analysis library (and goes beyond it in some cases), thanks to the efforts of Larry Paulson and Wenda Li (the former of whom ported huge amounts of HOL Light material to Isabelle/HOL).

Two things set my efforts apart from Harrison's: first, I aimed to provide structured Isar proofs that are (hopefully) more human-readable than the HOL Light tactic scripts. Second, Harrison often sought out more elementary versions of the proofs to reduce the amount of material required to be formalised. I, on the other hand, tried to develop all the library material beyond the minimum that was required for a particular end result. This can be seen e.g. with the Riemann ζ and the Dirichlet L functions. Harrison only defined $\zeta(s)$ for $\mathrm{Re}(s) > 0$ and $L(s)$ only at $s = 1$, whereas I defined them on the full complex plane and proved many additional advanced theorems about them.

This approach of building a large library of analytic number theory (and of course much infrastructure below it, most of it having been built by other people) turned proving the PNT and Dirichlet's Theorem from a big and daunting project into something that can be done in a few days without too much effort. Many of the proofs are also much shorter and more 'high-level' (most notably the proof of the non-vanishing of $\zeta(s)$ for $\mathrm{Re}(s) \geq 1$).

8.3 Further Work

Some time after the submission of the article, Rodrigo Raya (an undergraduate exchange student at the time) formalised Chapter 8 from Apostol's book mostly by himself under my supervision [83]. This chapter contains much material, including:

- periodic arithmetic functions and their finite Fourier expansions
- (generalised) Ramanujan sums
- Gauss sums and separable Dirichlet characters
- induced character moduli and primitive characters
- the Pólya–Vinogradov inequality

Additionally, I have since also formalised some more notable and non-trivial facts about the Riemann ζ function:

- its Laurent series expansion in terms of the Stieltjes constants [34]
- the Hadjicostas–Chapman formula involving Γ and ζ [21, 34, 51]
- Beukers' analytic version of Apéry's proof that $\zeta(3)$ is irrational [35]

None of these had been formalised before. All of these together took only about a week of casual work, which is surely a credit to how mature Isabelle's tools and libraries are for mathematical formalisation projects in this area. The proof of the irrationality of $\zeta(3)$ in particular was completed in about 2 or 3 days despite containing some rather nasty multidimensional integrals (which are always very painful to do rigorously due to all the integrability side conditions). In comparison, an earlier project to formalise the same theorem in Coq [23, 75] (following a non-analytic route and without having the PNT available) was a much more involved effort. However, the two projects are difficult to compare since the Coq formalisation concentrated more on developing specialised tools for *creative telescoping* and Apéry's theorem was more of an interesting application than the final goal to them.

The most interesting next step for this project would be to formalise more tools from modern number theory: fittingly, Apostol's book on analytic number theory is the first part of a series of two books, the second one being titled *Modular Functions and Dirichlet Series in Number Theory* [4]. This covers elliptic functions, modular functions, and modular forms. Formalising these would certainly be a natural, albeit ambitious next step. The related subject of elliptic curves, with their many applications both in abstract and computational number theory and in cryptography, would also be a very rewarding target for a formalisation project.

9 Concluding Remarks

“I rarely end up where I was intending to go,
but often I end up somewhere I needed to be.”

— Dirk Gently in *The Long Dark Tea-Time of the Soul* by Douglas Adams (1988)

The four publications reproduced in this dissertation only show a part of the entire picture. I would therefore like to use this last section to mention some related work that I have done but that has not made its way into a published article, or where that article did not quite fit the overall theme of this dissertation. Overall, I have written 49 entries in the *Archive of Formal Proofs* (some of them with collaborators), with a total of roughly 87,000 lines. Additionally, I have contributed about 24,000 lines of various material to the Isabelle distribution. I also took care of 18 of the 100 theorems on Freek Wiedijk's 'Top 100 Theorems to Formalise' list[1] in Isabelle (most of the remaining ones have been formalised in Isabelle by others already).

Apart from the material mentioned in Chapter 4, one very significant part of my work was the formalisation and analysis of classic textbook algorithms and data structures, especially probabilistic ones. Notable examples are:

- the Fisher–Yates shuffle [27]
- Treaps and Skip Lists (joint work with Max W. Haslbeck and Tobias Nipkow) [41, 60]
- randomised and deterministic QuickSort [38]
- median-of-medians selection [37]
- the $\Theta(n \log n)$ lower bound for comparison-based sorting algorithms [28]

The difficulty here was usually not in sophisticated mathematics but rather in how to best express the pen-and-paper arguments in the system; however, some mathematical results are required, e.g. Stirling's formula for $n!$ or the asymptotic growth of the harmonic numbers.

I also did some work on social choice theory research in collaboration with Felix Brandt and his group [16]. They used SMT solving to solve an open conjecture in randomised social choice, but referees were sceptical of this kind of proof. In particular, the SMT instance that was fed to the solver was generated by an unverified Java program, and the history of computer proofs (as briefly mentioned in the introduction) shows that such programs do often contain bugs. I therefore formalised the entire proof and the background theory that it relies on in Isabelle/HOL with the assistance of Brandt et al. It seems to me that this may well be a good way of increasing the trustworthiness of computer proofs such as these in general.

[1] `https://www.cs.ru.nl/~freek/100/`

Then, of course, there are numerous smaller things – too numerous and too small to mention them all.

Therefore, to conclude, I would like to turn to something that is perhaps more interesting to the reader: a few lessons and thoughts that I gathered during these six-ish years of formalising mathematics. The formalisation of mathematics has certainly come a long way since the days of Whitehead and Russell – I certainly doubt they could have imagined a formal proof of the Prime Number Theorem after their ordeal with even the most basic of mathematics in the *Principia*. However, it is still the case that formalising mathematics with a proof assistant is much more tedious and lengthy than on paper. I like to call this *The Curse of de Bruijn*, as de Bruijn noticed empirically that translating even 'very meticulous ordinary mathematics' into a formal proof in Automath resulted in a large constant-factor blowup, the constant of which has since been known as the *de Bruijn factor* (now also in systems other than Automath).

But why is this so? One of the many reasons – but a significant one – is that much of the work in an 'ordinary' pen-and-paper proof is, in some sense, externalised. First, it is externalised to previous mathematicians: a mathematician can, in principle, use any widely accepted theorem proven by any other person, even if they do not understand the proof. Certainly no one will require them to redo the entire proof by themselves before using it. The user of a proof assistant, however, will find that for any given concept or statement from the mathematical literature, there is a high likelihood that it has not been formalised yet – or perhaps it has, but not in *their* system. This highlights the importance of *good libraries* and *better interaction between systems*. The latter is not really my area of expertise, but I have it on good authority that people are working on such things at this moment, so perhaps I can focus on the former.

Second, work is often externalised to the reader. This already begins with notation: pen-and-paper mathematics often uses highly ambiguous notation and relies on context and the reader's intelligence to figure out what is actually meant. Furthermore, uninteresting side conditions (such as non-zeroness of a factor being cancelled) are routintely omitted in the presentation; sometimes even large chunks of a proof are omitted with the justification that the reader should readily be able to reconstruct them themselves based on the remaining parts. Some of this is surely beyond hope for automation in the foreseeable future, but there are also many small-scale examples of this. Many facts are obvious to a human due to domain-specific knowledge, but theorem provers typically have no built-in support for them, requiring input from the human. Examples of this are:

- $\sqrt{16}$ and $\sqrt{12}$ can readily be rewritten to 4 and $3\sqrt{2}$, respectively
- $x^2 - 2x - 2$ has the two real solutions $1 \pm \sqrt{3}$ by the quadratic formula taught in secondary school
- how to take a derivative
- how to expand a rational function into partial fractions
- $n \log n \in o(n^{1.5})$ since one factor of n can be cancelled and the remaining $\log n$ is eclipsed by the $n^{0.5}$
- $x^{-4}(1 - x^2/2 - \cos(x/(1 - x^2)))$ tends to $\frac{23}{24}$ for $x \longrightarrow 0$ by Taylor expansion

In my opinion, there is a great shortage of such *special-purpose mathematical automation* in theorem provers, and this is another issue that I would like to tackle. I believe that with `real_asymp` et al., I have at least made a valiant effort to begin such an undertaking.

On that note, I would like to end with what I intend to do now, and what I would like the rest of the theorem proving community (and the Isabelle community in particular) to do: Have fun! Formalise more mathematics! Build better tools! And, to adapt a quote by David Hilbert:

> From the paradise that Whitehead and Russell have created for us,
> no one shall be able to expel us!

Bibliography

[1] R. Affeldt, C. Cohen and D. Rouhling. ‘Formalization Techniques for Asymptotic Reasoning in Classical Analysis’. In: *Journal of Formalized Reasoning* 11.1 (2018), pp. 43–76. DOI: 10.6092/issn.1972-5787/8124.

[2] M. Akra and L. Bazzi. ‘On the Solution of Linear Recurrence Equations’. In: *Computational Optimization and Applications* 10.2 (1998), pp. 195–210. ISSN: 0926-6003. DOI: 10.1023/A:1018373005182.

[3] T. M. Apostol. *Introduction to analytic number theory*. Undergraduate Texts in Mathematics. Springer-Verlag, 1976. ISBN: 978-0-387-90163-3. DOI: 10.1007/978-1-4757-5579-4.

[4] T. M. Apostol. *Modular Functions and Dirichlet Series in Number Theory*. Vol. 41. Graduate Texts in Mathematics. Springer-Verlag, 1990, p. 207. ISBN: 978-0-387-97127-8. DOI: 10.1007/978-1-4612-0999-7.

[5] A. Asperti and W. Ricciotti. ‘A proof of Bertrand’s postulate’. In: *Journal of Formalized Reasoning* 5.1 (2012), pp. 37–57. ISSN: 1972-5787. DOI: 10.6092/issn.1972-5787/3406.

[6] J. Avigad and K. Donnelly. ‘A Decision Procedure for Linear “Big O” Equations’. In: *Journal of Automated Reasoning* 38.4 (2007), pp. 353–373. DOI: 10.1007/s10817-007-9066-1.

[7] J. Avigad et al. ‘A Formally Verified Proof of the Prime Number Theorem’. In: *ACM Trans. Comput. Logic* 9.1 (Dec. 2007). ISSN: 1529-3785. DOI: 10.1145/1297658.1297660.

[8] R. Bagnara, A. Zaccagnini and T. Zolo. *The Automatic Solution of Recurrence Relations. I. Linear Recurrences of Finite Order with Constant Coefficients*. Quaderno 334. Dipartimento di Matematica, Università di Parma, Italy, 2003. URL: http://www.cs.unipr.it/Publications.

[9] L. Bazzi and S. K. Mitter. ‘The Solution of Linear Probabilistic Recurrence Relations’. In: *Algorithmica* 36.1 (2003), pp. 41–57. DOI: 10.1007/s00453-002-1003-4.

[10] J. L. Bentley, D. Haken and J. B. Saxe. ‘A general method for solving divide-and-conquer recurrences’. In: *ACM SIGACT News* 12.3 (Sept. 1980), pp. 36–44. DOI: 10.1145/1008861.1008865.

[11] J. Biendarra and M. Eberl. ‘Bertrand’s postulate’. In: *Archive of Formal Proofs* (Jan. 2017). http://isa-afp.org/entries/Bertrands_Postulate.html, Formal proof development. ISSN: 2150-914x.

[12] J. Biendarra et al. *Defining (Co)datatypes and Primitively (Co)recursive Functions in Isabelle/HOL.* 2017. URL: `https://isabelle.in.tum.de/dist/doc/datatypes.pdf`.

[13] J. Biendarra et al. 'Foundational (Co)datatypes and (Co)recursion for Higher-Order Logic'. In: *Frontiers of Combining Systems – 11th International Symposium, FroCoS 2017.* Ed. by C. Dixon and M. Finger. Vol. 10483. Lecture Notes in Computer Science. Springer, 2017, pp. 3–21. DOI: `10.1007/978-3-319-66167-4_1`.

[14] R. Boyer, M. Kaufmann and J. Moore. 'The Boyer–Moore theorem prover and its interactive enhancement'. In: *Computers & Mathematics with Applications* 29.2 (1995), pp. 27–62. ISSN: 0898-1221. DOI: `10.1016/0898-1221(94)00215-7`.

[15] R. S. Boyer and J. S. Moore. *A computational logic.* ACM monograph series. Academic Press, 1979. ISBN: 9780121229504.

[16] F. Brandl et al. 'Proving the Incompatibility of Efficiency and Strategyproofness via SMT Solving'. In: *Journal of the ACM* 65.2 (Jan. 2018), 6:1–6:28. ISSN: 0004-5411. DOI: `10.1145/3125642`.

[17] L. Bulwahn. 'The New Quickcheck for Isabelle'. In: *Certified Programs and Proofs: Second International Conference, CPP 2012.* Ed. by C. Hawblitzel and D. Miller. Springer, 2012, pp. 92–108. ISBN: 978-3-642-35308-6. DOI: `10.1007/978-3-642-35308-6_10`.

[18] L. Bulwahn and M. Eberl. 'Bernoulli Numbers'. In: *Archive of Formal Proofs* (Jan. 2017). `http://isa-afp.org/entries/Bernoulli.html`, Formal proof development. ISSN: 2150-914x.

[19] M. Carneiro. *Arithmetic in Metamath, Case Study: Bertrand's Postulate.* 2015. arXiv: `1503.02349`.

[20] M. Carneiro. 'Formalization of the prime number theorem and Dirichlet's theorem'. In: *9th Conference on Intelligent Computer Mathematics (CICM 2016).* 2016, pp. 10–13. URL: `http://ceur-ws.org/Vol-1785/F3.pdf`.

[21] R. Chapman. *A proof of Hadjicostas's conjecture.* 2004. arXiv: `math/0405478`.

[22] J. Chen. *An Implementation of Homotopy Type Theory in Isabelle/Pure.* 2019. arXiv: `1911.00399`.

[23] F. Chyzak et al. 'A Computer-Algebra-Based Formal Proof of the Irrationality of $\zeta(3)$'. In: *Interactive Theorem Proving – 5th International Conference, ITP 2014.* Ed. by G. Klein and R. Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, 2014, pp. 160–176. DOI: `10.1007/978-3-319-08970-6_11`.

[24] The Coq Development Team. *The Coq Proof Assistant, version 8.10.0.* Oct. 2019. DOI: `10.5281/zenodo.3476303`.

[25] T. H. Cormen et al. *Introduction to Algorithms.* 2nd. McGraw-Hill Higher Education, 2001. ISBN: 0070131511.

[26] M. Eberl. 'Catalan Numbers'. In: *Archive of Formal Proofs* (June 2016). `http://isa-afp.org/entries/Catalan_Numbers.html`, Formal proof development. ISSN: 2150-914x.

[27] M. Eberl. 'Fisher–Yates shuffle'. In: *Archive of Formal Proofs* (Sept. 2016). `http://isa-afp.org/entries/Fisher_Yates.html`, Formal proof development. ISSN: 2150-914x.

[28] M. Eberl. 'Lower bound on comparison-based sorting algorithms'. In: *Archive of Formal Proofs* (Mar. 2017). `http://isa-afp.org/entries/Comparison_Sort_Lower_Bound.html`, Formal proof development. ISSN: 2150-914x.

[29] M. Eberl. 'Nine Chapters of Analytic Number Theory in Isabelle/HOL'. In: *Interactive Theorem Proving*. Leibniz International Proceedings in Informatics, 2019. DOI: `10.4230/LIPIcs.ITP.2019.16`.

[30] M. Eberl. 'Proving Divide and Conquer Complexities in Isabelle/HOL'. In: *Journal of Automated Reasoning* 58.4 (Apr. 2017), pp. 483–508. ISSN: 1573-0670. DOI: `10.1007/s10817-016-9378-0`.

[31] M. Eberl. 'Stirling's formula'. In: *Archive of Formal Proofs* (Sept. 2016). `http://isa-afp.org/entries/Stirling_Formula.html`, Formal proof development. ISSN: 2150-914x.

[32] M. Eberl. 'The Error Function'. In: *Archive of Formal Proofs* (Feb. 2018). `http://isa-afp.org/entries/Error_Function.html`, Formal proof development. ISSN: 2150-914x.

[33] M. Eberl. 'The Euler–MacLaurin Formula'. In: *Archive of Formal Proofs* (Mar. 2017). `http://isa-afp.org/entries/Euler_MacLaurin.html`, Formal proof development. ISSN: 2150-914x.

[34] M. Eberl. 'The Hurwitz and Riemann ζ Functions'. In: *Archive of Formal Proofs* (Oct. 2017). `http://isa-afp.org/entries/Zeta_Function.html`, Formal proof development. ISSN: 2150-914x.

[35] M. Eberl. 'The Irrationality of $\zeta(3)$'. In: *Archive of Formal Proofs* (Dec. 2019). `http://isa-afp.org/entries/Zeta_3_Irrational.html`, Formal proof development. ISSN: 2150-914x.

[36] M. Eberl. 'The Lambert W Function on the Reals'. In: *Archive of Formal Proofs* (Apr. 2020). `http://isa-afp.org/entries/Lambert_W.html`, Formal proof development. ISSN: 2150-914x.

[37] M. Eberl. 'The Median-of-Medians Selection Algorithm'. In: *Archive of Formal Proofs* (Dec. 2017). `http://isa-afp.org/entries/Median_Of_Medians_Selection.html`, Formal proof development. ISSN: 2150-914x.

[38] M. Eberl. 'The number of comparisons in QuickSort'. In: *Archive of Formal Proofs* (Mar. 2017). `http://isa-afp.org/entries/Quick_Sort_Cost.html`, Formal proof development. ISSN: 2150-914x.

[39] M. Eberl. ‘Verified Real Asymptotics in Isabelle/HOL’. In: *International Symposium on Symbolic and Algebraic Computation.* ISSAC ’19. ACM, 2019. ISBN: 978-1-4503-6084-5. DOI: 10.1145/3326229.3326240.

[40] M. Eberl. ‘Verified Solving and Asymptotics of Linear Recurrences’. In: *8th ACM SIGPLAN International Conference on Certified Programs and Proofs.* CPP 2019. ACM, 2019, pp. 27–37. ISBN: 978-1-4503-6222-1. DOI: 10.1145/3293880.3294090.

[41] M. Eberl, M. W. Haslbeck and T. Nipkow. ‘Verified Analysis of Random Binary Tree Structures’. In: *Interactive Theorem Proving – 9th International Conference, ITP 2018.* Ed. by J. Avigad and A. Mahboubi. Vol. 10895. Lecture Notes in Computer Science. Springer, 2018, pp. 196–214. DOI: 10.1007/978-3-319-94821-8_12.

[42] P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* 1st ed. Cambridge University Press, 2009. ISBN: 9780521898065.

[43] P. Flajolet, P. Zimmermann and B. Salvy. *Lambda-Upsilon-Omega: The 1989 cookbook.* Research Report RR-1073. INRIA, 1989. URL: https://hal.inria.fr/inria-00075486.

[44] G. Gonthier. ‘The Four Colour Theorem: Engineering of a Formal Proof’. In: *Computer Mathematics, 8th Asian Symposium, ASCM 2007. Revised and Invited Papers.* Ed. by D. Kapur. Vol. 5081. Lecture Notes in Computer Science. Springer, 2007, p. 333. DOI: 10.1007/978-3-540-87827-8_28.

[45] G. Gonthier et al. ‘A Machine-Checked Proof of the Odd Order Theorem’. In: *Interactive Theorem Proving – 4th International Conference, ITP 2013.* Ed. by S. Blazy, C. Paulin-Mohring and D. Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer, 2013, pp. 163–179. DOI: 10.1007/978-3-642-39634-2_14.

[46] M. J. C. Gordon. ‘Introduction to the HOL System’. In: *1991 International Workshop on the HOL Theorem Proving System and its Applications.* Ed. by M. Archer et al. IEEE Computer Society, 1991, pp. 2–3.

[47] M. J. C. Gordon, R. Milner and C. P. Wadsworth. *Edinburgh LCF.* Vol. 78. Lecture Notes in Computer Science. Springer, 1979. ISBN: 3-540-09724-4. DOI: 10.1007/3-540-09724-4.

[48] R. L. Graham, D. E. Knuth and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* 2nd. Addison–Wesley, 1994. ISBN: 0201558025.

[49] R. P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction.* Pearson Addison Wesley, 2004. ISBN: 9780201726343.

[50] D. Gruntz. ‘On Computing Limits in a Symbolic Manipulation System’. PhD thesis. ETH Zürich, 1996. DOI: 10.3929/ethz-a-001631582.

[51] P. Hadjicostas. *A conjecture-generalization of Sondow’s formula.* 2004. arXiv: math/0405423.

[52] F. Haftmann. 'Code Generation from Specifications in Higher Order Logic'. PhD thesis. Technische Universität München, 2009. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20091208-886023-1-1.

[53] F. Haftmann and T. Nipkow. 'Code Generation via Higher-Order Rewrite Systems'. In: *Functional and Logic Programming, 10th International Symposium, FLOPS 2010.* Ed. by M. Blume, N. Kobayashi and G. Vidal. Vol. 6009. Lecture Notes in Computer Science. Springer, 2010, pp. 103–117. DOI: 10.1007/978-3-642-12251-4_9.

[54] T. C. Hales. 'The Jordan Curve Theorem, Formally and Informally'. In: *The American Mathematical Monthly* 114.10 (2007), pp. 882–894. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/27642361.

[55] T. C. Hales et al. *A formal proof of the Kepler conjecture.* 2015. arXiv: 1501.02155.

[56] J. M. Han and F. van Doorn. 'A formal proof of the independence of the continuum hypothesis'. In: *9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020.* Ed. by J. Blanchette and C. Hritcu. ACM, 2020, pp. 353–366. DOI: 10.1145/3372885.3373826.

[57] J. Harrison. 'A formalized proof of Dirichlet's theorem on primes in arithmetic progression'. In: *Journal of Formalized Reasoning* 2.1 (2009), pp. 63–83. DOI: 10.6092/issn.1972-5787/1558.

[58] J. Harrison. 'Formalizing an analytic proof of the prime number theorem (Dedicated to Mike Gordon on the occasion of his 60th birthday)'. In: *Journal of Automated Reasoning* 43.3 (Oct. 2009), pp. 243–261. ISSN: 1573-0670. DOI: 10.1007/s10817-009-9145-6.

[59] J. Harrison. *Theorem proving with the real numbers.* CPHC/BCS distinguished dissertations. Springer, 1998. ISBN: 978-3-540-76256-0.

[60] M. W. Haslbeck and M. Eberl. 'Skip Lists'. In: *Archive of Formal Proofs* (Jan. 2020). http://isa-afp.org/entries/Skip_Lists.html, Formal proof development. ISSN: 2150-914x.

[61] M. J. H. Heule, O. Kullmann and V. W. Marek. 'Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer'. In: *Theory and Applications of Satisfiability Testing – SAT 2016.* Springer, 2016, pp. 228–245. DOI: 10.1007/978-3-319-40970-2_15.

[62] J. van der Hoeven. *Transseries and real differential algebra.* Vol. 1888. Lecture Notes in Mathematics. Springer-Verlag, 2006.

[63] J. Hölzl. 'Proving Inequalities over Reals with Computation in Isabelle/HOL'. In: *ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS'09).* Ed. by G. D. Reis and L. Théry. Aug. 2009, pp. 38–45.

[64] J. Hölzl, F. Immler and B. Huffman. ‘Type Classes and Filters for Mathematical Analysis in Isabelle/HOL’. In: *Interactive Theorem Proving – 4th International Conference, ITP 2013*. Ed. by S. Blazy, C. Paulin-Mohring and D. Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer, 2013, pp. 279–294. DOI: 10.1007/978-3-642-39634-2_21.

[65] F. Immler. ‘A Verified ODE Solver and the Lorenz Attractor’. In: *Journal of Automated Reasoning* 61.1-4 (2018), pp. 73–111. DOI: 10.1007/s10817-017-9448-y.

[66] F. Immler, J. Rädle and M. Wenzel. ‘Virtualization of HOL4 in Isabelle’. In: *10th International Conference on Interactive Theorem Proving, ITP 2019*. Ed. by J. Harrison, J. O’Leary and A. Tolmach. Vol. 141. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 21:1–21:18. DOI: 10.4230/LIPIcs.ITP.2019.21.

[67] R. M. Karp. ‘Probabilistic Recurrence Relations’. In: *Journal of the ACM* 41.6 (1994), pp. 1136–1150. DOI: 10.1145/195613.195632.

[68] D. E. Knuth. ‘Big Omicron and Big Omega and Big Theta’. In: *SIGACT News* 8.2 (Apr. 1976), pp. 18–24. ISSN: 0163-5700. DOI: 10.1145/1008328.1008329.

[69] A. Krauss. ‘Defining Recursive Functions in Isabelle/HOL’. 2010. URL: https://isabelle.in.tum.de/doc/functions.pdf.

[70] A. Krauss. ‘Partial and Nested Recursive Function Definitions in Higher-order Logic’. In: *Journal of Automated Reasoning* 44.4 (2010), pp. 303–336. DOI: 10.1007/s10817-009-9157-2.

[71] E. Landau. *Grundlagen der Analysis*. Akademische Verlagsgesellschaft, 1930.

[72] T. Leighton. ‘Notes on Better Master Theorems for Divide-and-Conquer Recurrences (MIT lecture notes)’. Lecture notes, MIT. 1996. URL: https://courses.csail.mit.edu/6.046/spring04/handouts/akrabazzi.pdf.

[73] W. Li and L. C. Paulson. ‘Evaluating Winding Numbers and Counting Complex Roots Through Cauchy Indices in Isabelle/HOL’. In: *Journal of Automated Reasoning* 64.2 (2020), pp. 331–360. DOI: 10.1007/s10817-019-09521-3.

[74] F. Luccio and L. Pagli. ‘On the Height of Height-Balanced Trees’. In: *IEEE Transactions on Computers* 25.1 (1976), pp. 87–91. DOI: 10.1109/TC.1976.5009208.

[75] A. Mahboubi and T. Sibut-Pinote. *A Formal Proof of the Irrationality of $\zeta(3)$*. 2019. arXiv: 1912.06611.

[76] D. Matthews. *The Poly/ML implementation of Standard ML*. URL: https://www.polyml.org/.

[77] P. McCorduck. *Machines who think: a personal inquiry into the history and prospects of artificial intelligence*. 2nd ed. A K Peters/CRC Press, 2004. ISBN: 9781568812052.

[78] W. McCune. ‘Solution of the Robbins Problem’. In: *Journal of Automated Reasoning* 19.3 (1997), pp. 263–276. DOI: 10.1023/A:1005843212881.

[79] A. Naumowicz and A. Kornilowicz. 'A Brief Overview of Mizar'. In: *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*. Ed. by S. Berghofer et al. Vol. 5674. Lecture Notes in Computer Science. Springer, 2009, pp. 67–72. DOI: 10.1007/978-3-642-03359-9_5.

[80] R. Nederpelt, J. Geuvers and R. de Vrijer. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1994. ISBN: 9780080887180.

[81] T. Nipkow, L. C. Paulson and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-43376-7. DOI: 10.1007/3-540-45949-9.

[82] M. Rau and T. Nipkow. 'Closest Pair of Points Algorithms'. In: *Archive of Formal Proofs* (Jan. 2020). http://isa-afp.org/entries/Closest_Pair_Points.html, Formal proof development. ISSN: 2150-914x.

[83] R. Raya and M. Eberl. 'Gauss Sums and the Pólya–Vinogradov Inequality'. In: *Archive of Formal Proofs* (Dec. 2019). http://isa-afp.org/entries/Gauss_Sums.html, Formal proof development. ISSN: 2150-914x.

[84] M. Riccardi. 'Pocklington's Theorem and Bertrand's Postulate'. In: *Formalized Mathematics* 14 (Jan. 2006), pp. 47–52. DOI: 10.2478/v10037-006-0007-y.

[85] D. Richardson. 'Some Undecidable Problems Involving Elementary Functions of a Real Variable'. In: *The Journal of Symbolic Logic* 33.4 (1968), pp. 514–520. ISSN: 00224812. DOI: 10.2307/2271358. URL: http://www.jstor.org/stable/2271358.

[86] D. Richardson. 'The Elementary Constant Problem'. In: *Papers from the International Symposium on Symbolic and Algebraic Computation*. ISSAC '92. ACM, 1992, pp. 108–116. ISBN: 0897914899. DOI: 10.1145/143242.143284.

[87] D. Richardson et al. 'Asymptotic Expansions of exp-log Functions'. In: *1996 International Symposium on Symbolic and Algebraic Computation*. ISSAC '96. ACM, 1996, pp. 309–313. ISBN: 0-89791-796-0. DOI: 10.1145/236869.237089.

[88] D. Russinoff. 'A Mechanical Proof of Quadratic Reciprocity'. In: *Journal of Automated Reasoning* 8 (Feb. 1992), pp. 3–21. DOI: 10.1007/BF00263446.

[89] B. Salvy and J. Shackell. 'Measured limits and multiseries'. In: *Journal of the London Mathematical Society* 82.3 (Oct. 2010), pp. 747–762. ISSN: 0024-6107. DOI: 10.1112/jlms/jdq057.

[90] J. R. Shackell. *Symbolic Asymptotics*. Vol. 12. Algorithms and Computation in Mathematics. Springer, 2004. ISBN: 3-540-21097-0.

[91] P. Singh. 'The so-called Fibonacci numbers in ancient and medieval India'. In: *Historia Mathematica* 12.3 (1985), pp. 229–244. ISSN: 0315-0860. DOI: 10.1016/0315-0860(85)90021-7.

[92] J. Tassarotti and R. Harper. 'Verified Tail Bounds for Randomized Programs'. In: *Interactive Theorem Proving – 9th International Conference, ITP 2018*. Ed. by J. Avigad and A. Mahboubi. Vol. 10895. Lecture Notes in Computer Science. Springer, 2018, pp. 560–578. DOI: 10.1007/978-3-319-94821-8_33.

[93] L. Théry. 'Proving Pearl: Knuth's Algorithm for Prime Numbers'. In: *Theorem Proving in Higher Order Logics*. Ed. by D. Basin and B. Wolff. Springer, 2003, pp. 304–318. ISBN: 978-3-540-45130-3. DOI: 10.1007/10930755_20.

[94] R. Thiemann and A. Yamada. 'Algebraic numbers in Isabelle/HOL'. In: *Interactive Theorem Proving: 7th International Conference, ITP 2016*. Ed. by J. C. Blanchette and S. Merz. Springer, 2016, pp. 391–408. DOI: 10.1007/978-3-319-43144-4_24.

[95] R. Thiemann and A. Yamada. 'Polynomial Factorization'. In: *Archive of Formal Proofs* (Jan. 2016). http://isa-afp.org/entries/Polynomial_Factorization.html, Formal proof development. ISSN: 2150-914x.

[96] L. van Benthem Jutting. 'Checking Landau's "Grundlagen" in the Automath system'. PhD thesis. Technische Hogeschool Eindhoven, 1977. DOI: 10.6100/IR23183.

[97] Waterloo Maple, Inc. *Maple 2019*. URL: https://www.maplesoft.com/products/maple/.

[98] M. Wenzel. 'Isabelle/jEdit as IDE for Domain-specific Formal Languages and Informal Text Documents'. In: *Electronic Proceedings in Theoretical Computer Science* 284 (Nov. 2018), pp. 71–84. ISSN: 2075-2180. DOI: 10.4204/eptcs.284.6.

[99] M. Wenzel. *The Isabelle/Isar Reference Manual*. Part of the Isabelle distribution. URL: https://isabelle.in.tum.de/dist/doc/isar-ref.pdf.

[100] M. Wenzel. 'Isabelle/Isar — a versatile environment for human-readable formal proof documents'. PhD thesis. Technische Universität München, 2002. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss2002020117092.

[101] F. Wiedijk. *The Seventeen Provers of the World, Foreword by Dana S. Scott*. Vol. 3600. Lecture Notes in Computer Science. Springer, 2006. ISBN: 3-540-30704-4. DOI: 10.1007/11542384.

[102] Wolfram Research, Inc. *Mathematica, version 12.0*. URL: https://www.wolfram.com/mathematica.

[103] B. Zhan and M. P. L. Haslbeck. 'Verifying Asymptotic Time Complexity of Imperative Programs in Isabelle'. In: *Automated Reasoning – 9th International Joint Conference, IJCAR 2018*. Ed. by D. Galmiche, S. Schulz and R. Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, 2018, pp. 532–548. DOI: 10.1007/978-3-319-94205-6_35.

Appendix

A Semi-Automatic Real Asymptotics

This chapter was originally published as an article in the proceedings of a peer-reviewed conference:

> M. Eberl.
> Verified Real Asymptotics in Isabelle/HOL.
> In *Proceedings of the International Symposium on Symbolic and Algebraic Computation.*
> ISSAC '19, New York, NY, USA, 2019. ACM.
> DOI: `10.1145/3326229.3326240`.

I am the sole author of this article, thus all contributions are mine.

Synopsis: This work provides a mostly automatic proof method in Isabelle/HOL that can handle various asymptotic properties of concrete real-valued functions built from standard operations such as arithmetic, exponentials, logarithms, trigonometric functions, etc. Similarly to the procedures used in computer algebra systems, it is based on Multiseries expansions (cf. the work by Shackell et al. [87]).

On the following pages, the full article is reproduced in its published form in accordance to the ACM author rights for reproduction in a dissertation. The official version in the ACM Digital Library can be found under the DOI cited above.

Verified Real Asymptotics in Isabelle/HOL

Manuel Eberl
Technical University of Munich
Garching b. München
manuel.eberl@tum.de

ABSTRACT

Interactive theorem provers (or *proof assistants*) are software with which mathematical definitions and theorems can be formalised. They assist the user in writing formal proofs and check the correctness of these proofs, typically down to the level of basic logical inference steps. This provides a very high degree of assurance that any proof accepted by them is actually sound. Theorem provers contain varying amounts of tools for automation to assist the user, but unlike computer algebra systems, their focus is not on efficient automatic computation.

In this work, we focus on the particular problem of symbolically determining and proving asymptotics of real-valued functions: limits, 'Big-O' statements, and asymptotic expansions. The tool that is presented here uses an approach based on multiseries expansions and can handle functions built from basic arithmetic and standard functions like exp, ln, sin, $|\cdot|$, etc. as well as parameters. The procedure is efficient enough to handle big problems and it is fully automatic in many cases.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software**.

KEYWORDS

asymptotics; interactive theorem proving; proof assistant; Isabelle; symbolic computation; real analysis

ACM Reference Format:
Manuel Eberl. 2019. Verified Real Asymptotics in Isabelle/HOL. In *International Symposium on Symbolic and Algebraic Computation (ISSAC '19), July 15–18, 2019, Beijing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3326229.3326240

1 INTERACTIVE THEOREM PROVERS

An *interactive theorem prover* (or *proof assistant*) is a piece of software designed to assist in the development of a *formal proof.* All definitions and proof steps have to be made in a formal way and proofs are checked by the computer – typically down to the level of basic logical inference. Consequently, this involves much more work than a paper proof, but also provides much more clarity and a high assurance that the proofs are actually sound.

There are many different theorem provers; some popular ones are Coq, Isabelle, HOL Light, HOL4, Lean, and Mizar. They differ in their underlying logic and in the infrastructure they provide. In this paper, we focus on *Isabelle/HOL*, which is the combination of the generic *Isabelle* theorem prover with *Higher-Order Logic.*

Unlike computer algebra systems, theorem provers focus not on providing a framework for computation designed to automatically return a result quickly, but on providing a consistent logical infrastructure in which mathematical definitions and proofs can be made with high assurance of correctness. Like many other systems, Isabelle has a *kernel*, which is the only part of the system that can produce theorems. This kernel provides only basic logical inference rules (such as *modus ponens* or ∀-introduction/elimination) and a mechanism for non-recursive definitions. Outside the kernel, many additional tools exist, such as *tactics* to automate proofs (e. g. rewriting, first-order logic, linear arithmetic) and more sophisticated definitional mechanisms (e. g. for recursive functions, inductive predicates, and algebraic datatypes). However, all these additional tools need to go through the kernel to prove a theorem or define a function. The intention behind this design is that:

(1) A user must not be able to accidentally introduce inconsistencies by defining e. g. $f(x) = f(x) + 1$.
(2) Bugs in proof automation or definitional tools (e. g. termination checkers for recursive definitions) must not compromise the integrity of the entire system.

Formal proofs – even computer-assisted ones – are very verbose, since every step of a proof has to be written down in much more detail than in a pen-and-paper proof. Many steps that are elided in paper proofs need to be written out explicitly. To make theorem provers usable for the formalisation of non-trivial mathematics, it is therefore crucial that they provide good proof automation to reduce the burden on the user.

This work focuses on the particular problem of proving asymptotic properties of concrete real-valued functions such as that in Figure 1. Computer algebra systems like Mathematica and Maple are very good at solving these problems. They employ specialised algorithms to compute asymptotic expansions for a wide variety of functions efficiently and fully automatically. On the other hand, to our knowledge, no theorem prover has anything comparable to this so far. In this work, we attempt to integrate a procedure very much like the one used by computer algebra systems into Isabelle/HOL with the goal of providing a similarly effective and automatic 'push-button' solution to prove limits and other asymptotic properties.

It is important to emphasise that, naturally, we did not invent any of the underlying methods used in this work (such as multiseries and expansions of exp-log functions [8, 12]). The novel contribution of our work is that we integrate these methods into a theorem prover in a way that is effective and convenient to use, and that we successfully use this for non-trivial mathematical developments.

As of Isabelle 2018, the work described here can be found in the `HOL-Real_Asymp` session, which is a part of the Isabelle distribution.

2 MOTIVATION

Proving limits in a theorem prover can be very tedious. Of course, simple limits like those of $\exp(1/x)$ or $1 - \frac{1}{x} + \ln x$ for $x \longrightarrow \infty$ can be proven in an entirely syntax-directed way, which makes them very easy to automate. Indeed, just passing the right theorems to Isabelle's general-purpose proof automation is enough to prove these two examples:

```
lemma filterlim (λx. exp (1/x)) (nhds 1) at_top
  by (force intro: tendsto_eq_intros real_tendsto_divide_at_top filterlim_ident)

lemma filterlim (λx. 1 − 1/x + ln x) at_top at_top
  by (force intro: tendsto_diff filterlim_tendsto_add_at_top
                   real_tendsto_divide_at_top filterlim_ident ln_at_top)
```

Even for these simple examples, one can see that one has to search for (or remember) many facts and pass them to the proof method, which a user might expect the system to do for them.

To make matters worse, the above problems were actually cherry-picked: If one permutes the summands in the second example a bit, the method does not work anymore, since many of the theorems only exist in one form; e. g. the rule *filterlim_tendsto_add_at_top* states that if $f(x) \longrightarrow c$ and $g(x) \longrightarrow \infty$, then $f(x) + g(x) \longrightarrow \infty$. However, there is no equivalent theorem for the situation of $g(x) + f(x)$ or $f(x) - g(x)$, or when $g(x) \longrightarrow -\infty$. When one has one of these situations, one first has to rearrange terms in order for the theorem to apply, which can become very tedious especially for larger terms. Proving different forms of these theorems for every possible situations would possibly improve this, but creating this would require a lot of effort and duplication.

Moreover, in many interesting examples, the proofs are *not* entirely syntax-directed. Consider the following examples:

$$\lim_{x\to\infty} \frac{x+1}{x-1} \qquad \lim_{x\to\infty} \frac{x}{(\ln x)^c} \qquad \lim_{x\to 0} \frac{\sin(\tan x) - \tan(\sin x)}{x^7}$$

These are easy to solve by hand:

- The first one can be rewritten to $(1 + x^{-1})/(1 - x^{-1})$, for which the syntax-directed method immediately gives us the limit 1.
- The second one can be attacked by applying L'Hospital's rule again and again until we get $C_1 x (\ln x)^{C_2}$ with $C_1 > 0$ and $C_2 \geq 0$, which clearly goes to ∞.
- The third one can be solved using Taylor expansions to find that the limit is $-\frac{1}{30}$.

The drawback of the first two methods is that they typically require a certain amount of creativity and that there is no clear-cut class of problems on which they will work; the third method can become very tedious and error-prone on paper. In Isabelle, *all* of these methods are typically tedious, since 'obvious' steps need to be made explicit:

- Rewriting limits often requires proving side conditions; e. g. non-zeroness in the above example.
- L'Hospital's rule requires proving several other (albeit usually easier) limits and derivatives. Also, there are many different cases and each one requires a different variant of the rule to be used.
- Manual Taylor expansion-based proofs also require proving many derivatives and asymptotic estimates.

As one of the more extreme examples of the limit problems that can arise, consider Figure 1. This problem is part of Leighton's proof of the *Akra–Bazzi* theorem. The first formal proof of this statement in Isabelle [3] was 700 lines long and required considerable effort.

The above examples should demonstrate vividly that in order to do non-trivial mathematics involving asymptotic analysis in a theorem prover, better automation is needed. This state of affairs seems particularly bizarre when compared to the capabilities of modern computer algebra systems, which can typically solve problems like the ones above fully automatically within fractions of a second. The goal of this work is to bring Isabelle's capabilities w. r. t. limit computations and asymptotic analysis closer to that of a computer algebra system – while still constructing a machine-checked proof.

3 ASYMPTOTICS IN ISABELLE/HOL

First, we must explain some notation concerning asymptotics in Isabelle/HOL. Asymptotics in Isabelle are centred around *filters* [2]. For our purposes, a filter can be thought of as a kind of local neighbourhood or approach: The neighbourhood of a real number (i. e. all numbers that are sufficiently close to that number) is a filter, and so are ∞ (all sufficiently large numbers) and $-\infty$. There are also filters for $\pm\infty$ (all numbers whose absolute value is sufficiently large) and 0^+ (all sufficiently small positive numbers).

What is convenient about filters for theorem proving is that they compose in very natural ways and they can be used to uniformly express many different concepts related to topology, analysis, measure theory, or asymptotics: properties that hold 'eventually' ('for all values that are sufficiently …'), limits, summable families, pointwise and uniform continuity, derivatives, Landau symbols, etc.

For a detailed introduction to the way filters are used in Isabelle and the precise definition, see the paper by Hölzl *et al.* [6]. For this presentation, the notation listed in Table 1 will suffice. Similarly to the Isabelle notation, we will use $\underset{\infty}{\forall} x.\ P(x)$ from now on to denote that $P(x)$ holds for large enough x.

4 MULTISERIES

For the sake of simplicity, we only ever consider functions at the neighbourhood of ∞ and reduce all other cases to this. Therefore, any 'eventually', limit, or 'Big-O' from now on is to be understood for $x \longrightarrow \infty$ unless otherwise indicated.

At the core of our method is the concept of a *multiseries*. This is a representation of a Poincaré expansion of a given real-valued function, where each summand in the expansion is a monomial of the form $c b_1(x)^{e_1} \cdot \ldots \cdot b_k(x)^{e_k}$ with $c, e_1, \ldots, e_k \in \mathbb{R}$. The b_i are *basis functions*; they are functions tending to ∞ and they are the same for all summands. The list of the b_i is assumed to be ordered by descending growth in such a way that $\ln b_{i+1}(x) \in o(\ln b_i(x))$ for all i. The reason for this is that then $b_{i+1}(x)^e \in o(b_i(x)^{e'})$ for any $e, e' \in \mathbb{R}$ with $e' > 0$ so that we can easily compare two monomials asymptotically by comparing their exponent vectors

$$\lim_{x\to\infty}\left(1-\frac{1}{b\log^{1+\varepsilon}x}\right)^p\left(1+\frac{1}{\log^{\varepsilon/2}\left(bx+x\log^{-1-\varepsilon}x\right)}\right)-\left(1+\frac{1}{\log^{\varepsilon/2}x}\right)=0^+$$

for real-valued parameters b, p, ε with $b \in (0;1)$ and $\varepsilon > 0$

Figure 1: A limit-problem related to Leighton's proof of the Akra–Bazzi theorem.

Notation	Meaning
nhds c	neighbourhood filter around $c \in \mathbb{R}$
at_top	neighbourhood of ∞ in $\mathbb{R}$
$\forall x$ in F. $P(x)$	$P(x)$ holds *eventually* at F, i. e. for all x that are sufficiently F
filterlim f F G	$f(x) \xrightarrow{x \to G} F$
$f \in O_{[F]}(g)$	$\exists c{>}0.\ \forall x$ in F. $\lvert f(x)\rvert \le c\,\lvert g(x)\rvert$
$f \in o_{[F]}(g)$	$\forall c{>}0.\ \forall x$ in F. $\lvert f(x)\rvert \le c\,\lvert g(x)\rvert$
$f \sim_{[F]} g$	$f - g \in o_{[F]}(g)$

f and g are real-valued functions; F and G are filters.
If the filter argument $_F$ for the Landau symbols is not given, it defaults to *at_top*.

Table 1: Common asymptotic notation in Isabelle.

lexicographically. We call such a list a *well-formed basis.* A typical basis would be

$$\big(\exp(x\ln x),\ \ \exp(x),\ \ x,\ \ \ln x,\ \ \ln\ln x\big).$$

Abstractly, a multiseries is a formal power series in k variables standing for the basis functions, i. e. a function mapping exponent tuples $(e_1, \ldots, e_k) \in \mathbb{R}^k$ to coefficients in $\mathbb{R}$. We write this suggestively as

$$\sum C_{e_1,\ldots,e_k} b_1(x)^{e_1} \ldots b_k(x)^{e_k}\ .$$

The link between these formal objects and a concrete real function $f(x)$ is made by demanding that the multiseries be a Poincaré expansion of $f(x)$, i. e. that approximating $f(x)$ by the finite sum of all the monomials with exponent vectors $\geq_{\text{lex}} \bar{e}$ for some fixed $(\bar{e}_1, \ldots, \bar{e}_k)$ yields an error that is $o(b_1(x)^{\bar{e}_1} \ldots b_k(x)^{\bar{e}_k})$. Given a multiseries expansion of a function, its limit can then be computed by determining the leading monomial, i. e. the smallest exponent vector $(e_1, \ldots, e_k)$ whose coefficient is non-zero.

A computationally convenient view of multiseries is the following: By isolating the first basis element, we can view a multiseries as a (univariate) formal series in $b_1(x)$ whose coefficients are multiseries w. r. t. the $k-1$ basis functions $b_2(x), \ldots, b_k(x)$. This univariate series can be represented as a (possibly infinite) list whose elements are a pair of a coefficient (which is again a multiseries, but in $k-1$ variables) and an exponent (which is just a real number). Iterating this yields a kind of 'nested infinite list' structure with k levels.

The reason for this nested structure is the following: If one were to view multiseries simply as linear sequences of summands, cancellation can lead to an infinite number of zeros in the front and therefore to non-termination when trying to find the leading term. The nested list representation solves this problem because we can discard infinitely many zeros at once: Each element of the outermost list corresponds to a summand $c(x)b_1(x)^e$ with a multiseries expansion for $c(x)$ in terms of $b_2(x), \ldots, b_k(x)$ attached to it. If $c(x)$ is identically zero, we can discard the element and proceed with the next one until we find a term such that $c(x)$ is not identically zero. We then proceed analogously with the multiseries expansion of $c(x)$ etc. We refer to this process as *trimming* the representation of the multiseries.

When the distinction is necessary, we will write multiseries in double square brackets to distinguish them from the functions of which they are an expansion and separate the monomials with a vertical bar, e. g. $[\![x \mid 1]\!]_{(x,\ln)}$ is a multiseries w. r. t. the basis $(x, \ln)$ that corresponds to the formal sum $1 \cdot x^1 \ln(x)^0 + 1 \cdot x^0 \ln(x)^0$. Similarly, the notation $[\![cb(x)^e \mid \mathcal{F}]\!]_{b\#bs}$ indicates that the leading entry in the list has coefficient $[\![c]\!]_{bs}$ and exponent $e \in \mathbb{R}$ and $[\![\mathcal{F}]\!]_{b\#bs}$ is the remainder of the series. Variables denoting multiseries are in upper-case calligraphic font, e. g. $\mathcal{F}$. We also write addition, multiplication, etc. on multiseries as $\oplus$, $\odot$, etc. to distinguish them from the operations on real numbers.

To obtain nice theoretical properties of these multiseries, further restrictions must be placed on the support of this coefficient function (e. g. well-ordered or 'grid-based' [14]), but for our purposes, such assumptions need not be modelled explicitly. Since all our definitions work directly on the nested list representations, our multiseries have well-ordered coefficient support by construction. In fact, all the multiseries that our constructions yield will even be grid-based, but we do not need to talk about these notions in Isabelle/HOL explicitly to show the soundness of the approach; they are only relevant for a discussion of completeness.

Although the actual implementation in Isabelle is different due to lack of dependent types, we will, for the sake of this presentation, pretend that there exists a type of multiseries *mseries* that takes a list of basis function as its type parameter. Morally, we then have

$$[]\ \text{mseries} = \mathbb{R}$$
$$(b\ \#\ bs)\ \text{mseries} = (bs\ \text{mseries} \times \mathbb{R})\ \text{llist}$$

where α *llist* ('lazy list') is the type of possibly infinitely long lists with elements of type α and # denotes prepending a single basis function b to a list of functions bs. The *llist* type is a *codatatype* defined using Isabelle's codatatype package [1]. This allows us to define and reason about 'infinitely recursive' functions on infinite structures in an intuitive way. We make extensive use of this to define operations on multiseries, to link these formal series to real functions, and to prove correctness of the former w. r. t. the latter.

A technical subtlety that is not shown here is that every Multiseries implicitly 'knows' what function it is supposed to be an expansion of. For a Multiseries $\mathcal{F}$, we denote this function as $\mathcal{F}(x)$. For $\mathcal{F}$ to be *well-formed*, it needs to be a Poincaré expansion of $\mathcal{F}(x)$.

5 TRIMMING AND RECOGNISING ZERO

An important auxiliary operation on our multiseries representation is the afore-mentioned *trimming*. We call a multiseries *trimmed* if its leading monomial is non-zero. The process of *trimming* is to discard terms at the beginning of the list until this is the case. The difficulty here is that when considering a leading term of the form $[\![Cb(x)^e]\!]_{b\#bs}$, we need to decide whether or not it is identically 0 and discard it if it is; if we fail to recognise this and try to trim C instead, we may trim infinitely many zeros and never terminate.

Note, however, that the leading zeros can already contain some information: if e. g. $f(x) \sim 0 \cdot x^2 + \ldots$, then we can immediately see that $f(x) \in O(x^2)$. It is therefore sometimes sufficient to only partially trim a series. However, if we want more precise asymptotic information (in particular an asymptotic lower bound), we indeed need to make sure that the expansion is fully trimmed first. We will also see that some of the operations we will define on multiseries only work if the series is already trimmed. Trimming will therefore be an essential part in our expansion algorithm.

This leads us to the problem of recognising zeros, which is one of the central problems in automated symbolic asymptotics. Given some expression representing a real constant, how do we determine whether or not it is zero? Depending on the class of expressions considered, this problem ranges from difficult to undecidable [4]. For exp-log constants, it is known to be semi-decidable but the theory behind the decision procedure is fairly complicated and out of the scope of this work. [7]

We therefore chose a very simple approach: The algorithm uses a modular *zeroness oracle*[1] that receives an Isabelle term c of type $\mathbb{R}$ as an input and either fails with an informative error message or returns a theorem stating that $c = 0$, $c \geq 0$, $c > 0$ etc. depending on the exact configuration. A similar *eventual-zeroness oracle* receives a term f of type $\mathbb{R} \to \mathbb{R}$ and either fails silently or returns the theorem that $f(x)$ is eventually zero.

The standard implementation of these oracles uses Isabelle's *simplifier*, which is one of Isabelle's default proof methods. It is a simple directed term rewriting engine with a large setup of rewrite rules and some additional specialised procedures (e. g. for arithmetic). Since most zeroness problems encountered in practice are trivial (e. g. '$1 + 2 \cdot (1 + 0) = 0$'), this works reasonably well. This oracle is even able to handle parameters, but it sometimes reaches its limits when larger arithmetic expressions or functions like $\sqrt{\ }$, exp, and ln are involved. The user can load an optional additional oracle based on interval arithmetic approximation [5] that can handle many of these cases. Improving Isabelle's automation for problems such as these or adding more zeroness oracles is a worthwhile goal, but certainly outside the scope of this work. If the oracle fails, the users receive an error message indicating on which expression it got stuck so that they can provide a manual proof of its sign and re-run the proof method with this new knowledge.

It should be mentioned that the trimming algorithm sketched here only terminates if the eventual-zeroness oracle never fails on a provable result and that we do not encounter an all-zero expansion of a non-zero function. The latter would be an instance where we have some function $f(x)$ that goes to 0 faster than can be measured by the basis of its multiseries (e. g. $\exp(-x)$ with the basis (x)). It is the responsibility of the expansion algorithm to ensure that this does not happen. For the basic class of exp-log functions, this is ensured since the expansion algorithm always produces *convergent* expansions. When we add more functions (e. g. Γ), this is no longer the case and the analysis becomes more complicated. Whether or not the algorithm is still complete in such cases is not clear to us and we consider this beyond the scope of this work since, in the context of a proof method in an interactive theorem prover, completeness is desirable but not absolutely necessary.

Neither of these issues make our procedure untrustworthy: non-termination of the trimming will simply lead to non-termination of the entire algorithm. If the algorithm *does* terminate, a full proof will have been produced and has passed through the Isabelle kernel. The afore-mentioned issues can therefore, by design, only compromise the *completeness* of the algorithm, not its *soundness*.

6 OPERATIONS ON MULTISERIES

We will now give examples of how concrete operations on our multiseries expansion can be implemented. The presentation is fairly close to that in Isabelle but with simplified, type-theory inspired syntax. Due to space constraints, we only show a few basic operations and refer to Shackell [12] for the remaining ones.

6.1 Basic Arithmetic

Constant functions, the identity function and powers thereof have obvious multiseries representations. Also, given an abstract multiseries (i. e. a mapping from exponent vectors to coefficients), it is easy to see how they can be negated, added, and multiplied. As an example, negation and addition can be defined like this:

```
$(\ominus) :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries}$
$\ominus_{[]}\ [\![c]\!] = [\![-c]\!]$
$\ominus_{b\#bs}\ [\![Cb(x)^e \mid \mathcal{F}]\!] = [\![(\ominus_{bs} C)b(x)^e \mid \ominus_{b\#bs} \mathcal{F}]\!]$

$(\oplus) :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries} \to bs\ \text{mseries}$
$[\![c_1]\!] \oplus_{[]} [\![c_2]\!] = [\![c_1 + c_2]\!]$
$[\![C_1 b(x)^{e_1} \mid \mathcal{F}]\!] \oplus_{b\#bs} [\![C_2 b(x)^{e_2} \mid \mathcal{G}]\!] =$
    **if** $e_1 > e_2$ **then**
        $[\![C_1 b(x)^{e_1} \mid \mathcal{F} \oplus_{b\#bs} [\![C_2 b(x)^{e_2} \mid \mathcal{G}]\!]]\!]$
    **else if** $e_1 < e_2$ **then**
        $[\![C_2 b(x)^{e_2} \mid [\![C_1 b(x)^{e_1} \mid \mathcal{F}]\!] \oplus_{b\#bs} \mathcal{G}]\!]$
    **else**
        $[\![([\![C_1]\!] \oplus_{bs} [\![C_2]\!])\, b(x)^{e_1} \mid \mathcal{F} \oplus_{b\#bs} \mathcal{G}]\!]$
```

[1]Note that the name 'oracle' does not mean that we blindly trust the oracle. It is still required to return an Isabelle *theorem*.

In the addition algorithm, the first equation is the base case for an empty basis; the second equation 'merges' two series by descending exponents. Note that each of the three cases in the second equation contains a corecursive call to the addition function on the same basis and the third case additionally contains a recursive call to the addition function for the truncated basis.

Multiplication can be implemented analogously (using an auxiliary function that performs multiplication with a monomial $Cb(x)^e$), but operations like the division and powers are slightly more complicated: We first need to implement substitution of a multiseries into a asymptotic or convergent power series.

6.2 Substituting into a Power Series

For this, consider some function $h : \mathbb{R} \to \mathbb{R}$ that has an asymptotic power series expansion $c_0 + c_1 x + c_2 x^2 + \ldots$ at $x = 0$. Other points are, of course, also possible (including at ∞), but $x = 0$ will be enough for our purposes. If we have a function $f : \mathbb{R} \to \mathbb{R}$ with some multiseries expansion $\mathcal{F}$ whose leading exponent is negative, then f clearly tends to 0 for $x \to \infty$ and we can substitute the multiseries expansion for f into the power series expansion for h to obtain a multiseries expansion for $h(f(x))$. This can be implemented as follows:

$$\begin{aligned}
&\text{powser} :: \forall bs :: \text{basis}.\ \mathbb{R}\ \text{llist} \to bs\ \text{mseries} \to bs\ \text{mseries}\\
&\text{powser}\ []\ \mathcal{F} = [\![]\!]_{bs}\\
&\text{powser}\ [c \mid cs]\ \mathcal{F} = [\![c \mid \mathcal{F} \odot \text{powser}\ cs\ \mathcal{F}]\!]_{\text{bs}}
\end{aligned}$$

In particular, this allows us to turn a multiseries expansion for f into a multiseries expansion for $h \circ f$ if $f(x) \longrightarrow 0$ and h is analytic at 0. This will be a key ingredient for the remaining operations that we implement.

6.3 Division and Powers

Consider the multiplicative inverse function $x \to \frac{1}{x}$, which we can use to handle division. Given a function f with a multiseries expansion $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}$, we consider the remainder after dropping the leading term of the expansion, i. e. $\bar{f}(x) := f(x) - C(x)b(x)^e$ with the expansion $\bar{\mathcal{F}}$. We will use $\bar{\mathcal{F}}$ and $\bar{f}(x)$ with that meaning from now on. We can then write:

$$\frac{1}{f(x)} = \frac{1}{C(x)} b(x)^{-e} \frac{1}{1 + \frac{1}{C(x)} b(x)^{-e} \bar{f}(x)}$$

Since $t \to \frac{1}{1+t}$ has the power series expansion $1 + t + t^2 + \ldots$ at $t = 0$, we can then define the multiplicative inverse $\mathcal{I}(\mathcal{F})$:

$$\begin{aligned}
&\mathcal{I} :: \forall bs :: \text{basis}.\ bs\ \text{mseries} \to bs\ \text{mseries}\\
&\mathcal{I}([\![c]\!]_{[]}) = [\![\tfrac{1}{c}]\!]_{[]}\\
&\mathcal{I}([\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}) = [\![\mathcal{I}(C)b(x)^{-e}]\!] \odot_{b\#bs}\\
&\qquad\qquad \text{powser}\ [1, 1, \ldots]\ ([\![\mathcal{I}(C)b(x)^{-e}]\!] \odot_{b\#bs} \bar{\mathcal{F}})
\end{aligned}$$

Note that $\mathcal{I}$ is only well-defined if the multiseries it is given is trimmed. Also note that the argument of *powser* indeed always has a negative leading exponent here since *lead_exp*$(\bar{\mathcal{F}}) < e$.

The same approach can be used to handle the case $f(x)^u$ for any constant $u \in \mathbb{R}$ by writing

$$f(x)^u = c(x)^u b(x)^{ue} (1 + c(x)^{-1} b(x)^{-e} \bar{f}(x))^u$$

and using the power series expansion for $t \to (1 + t)^u$. Here the condition is that the multiseries must be trimmed with positive leading coefficient (so that it is eventually positive).

6.4 Sine and Cosine

The sine and cosine functions can also be treated similarly: Given $f(x) \sim \mathcal{F}$ with $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]_{b\#bs}$, we distinguish three cases:

- If $e < 0$, we substitute $\mathcal{F}$ into the power series for sin or cos.
- If $e > 0$, then $f(x)$ tends to infinity and no multiseries expansion for $\sin f(x)$ or $\cos f(x)$ can exist.
- If $e = 0$, we can write

  $$\sin f(x) = \sin C(x) \cdot \cos \bar{f}(x) + \cos C(x) \cdot \sin \bar{f}(x)$$

  and analogously for $\cos f(x)$. Expansions for $\sin C(x)$ and $\cos C(x)$ can be computed by a recursive call, and $\cos \bar{f}(x)$ and $\sin \bar{f}(x)$ are covered by the $e < 0$ case.

6.5 Logarithm

If we apply this same approach to the ln function, we arrive at the expression

$$\ln f(x) = \ln c(x) + e \ln b(x) + \ln\left(1 + c(x)^{-1} b(x)^{-e} \bar{f}(x)\right).$$

The first and last summand can be handled analogously to the previous cases; however, there is a problem in the second summand: If $e \neq 0$, we need a multiseries expansion for $\ln b(x)$ w. r. t. the basis $b\ \#\ bs$. In our definition of a well-formed basis, we assumed that for $i < n$, each $\ln b_i(x)$ has a known expansion in terms of $b_{i+1}, \ldots, b_n$ so that we only have a problem if b is the very last element in the basis (i. e. $bs = []$). The solution in this case is to add $b_{n+1}(x) := \ln b(x)$ as a new basis element at the very end of the basis. This ensures that when we now reach the last basis element b_{n+1}, the exponent e is zero and we do not encounter the problem. Note that inserting a new basis element is a *global* operation, which means that we must lift all expansions computed before to the new, larger basis. E. g. when we expand $f(x) + g(x)$ by first expanding $f(x) \sim \mathcal{F}$ and then $g(x) \sim \mathcal{G}$, the latter step may have enlarged the basis, in which case we need to lift $\mathcal{F}$ to the new basis.

6.6 Exponentials

For the exponential function, the situation is much more complicated. If $e < 0$, we can simply use the power series expansion for exp. If $e = 0$, we have $\exp(f(x)) = \exp(c(x)) \exp(\bar{f}(x))$ and have therefore reduced the problem to a recursive call and the '$e < 0$' case. If, on the other hand, $e > 0$, various case distinctions are required to determine whether $\exp(f(x))$ has an expansion w. r. t. the current basis or whether $\exp(f(x))$ or $\exp(-f(x))$ or some variation thereof has to be added as a new basis element – and if yes, where. For the details of this case distinction, we refer again to Shackell [12].

6.7 Other Functions

All functions discussed so far had sufficiently nice properties w. r. t. addition or multiplication of their argument that allowed us to handle the case $f(C(x) + \bar{f}(x))$. Indeed, this is enough to handle the class of all functions built from basic arithmetic, exp, ln as well as sin, cos, and tan at finite points. For functions without these nice properties, this simple approach does not work, as we will see later in section 9.

7 CONNECTING SERIES TO FUNCTIONS

We have defined a representation for multiseries and implemented a number of operations on them; however, we have so far not formally defined what it means for a multiseries to be an expansion of a particular function. Since we want to prove theorems inside the logic, this connection has to be defined explicitly inside the logic, and the correctness of all the operations we defined must be formally proven w. r. t. it. This is the crucial difference to Computer Algebra Systems and it results in a high level of confidence in the results produced by the expansion algorithm, but also includes a much greater amount of effort.

We will introduce a predicate $\mathrm{wf}_{bs}(\mathcal{F})$ whose meaning is that $\mathcal{F}$ is a well-formed multiseries w. r. t. the basis bs and it is a valid expansion of the function $\mathcal{F}(x)$ that it is implicitly connected to. The former includes things like 'the exponents are strictly decreasing' and that the depth of the nested list structure is the same as the length of the basis. To understand what the latter means explicitly, we recall the notion of a Poincaré expansion: If we take a finite initial segment of the (possibly infinite) series, we obtain an approximation to the function such that the error is 'Big-O' of the first omitted term. However, we will use a slightly different formulation of this that is more suitable for our representation of multiseries.

Recall that the type of multiseries is defined by recursion on the associated basis: A multiseries C w. r. t. the empty Basis is simply $[\![c]\!]_{[]}$ for a real constant c. The obvious definition of well-formedness in this case is to require that $C(x) = c$ eventually.

On the other hand, a multiseries w. r. t. a non-empty basis $b \# bs$ is a (possibly infinite) list. As mentioned before, these lists are a *codatatype* and a natural way to define predicates for a codatatype is *coinductively*. For this, we need to express $\mathrm{wf}_{b\#bs}([\![\mathcal{F}]\!])$ in terms of $\mathrm{wf}_{b\#bs}(\bar{\mathcal{F}})$, where $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]$. To see how to do this, it is instructive to consider the situation for a power series:

$$f(x) \sim cx^e + \mathcal{F} \longleftrightarrow f(x) \in O(x^e) \ \wedge\ e > \mathrm{lead\_exp}(\mathcal{F}) \ \wedge$$
$$f(x) - cx^e \sim \mathcal{F}$$

Adapting this for multiseries, we find that $\mathrm{wf}_{b\#bs}(\mathcal{F})$ requires:

$$\mathrm{wf}_{bs}(C) \ \wedge\ \mathrm{wf}_{b\#bs}(\bar{\mathcal{F}}) \ \wedge$$
$$e > \mathrm{lead\_exp}(\bar{\mathcal{F}}) \ \wedge\ \forall e'{>}e.\ \mathcal{F}(x) \in o(b(x)^{e'})$$

Since in our formalisation, multiseries can also have finite length, we also need to consider the case of an empty multiseries. There are two natural possibilities here:

(1) demand that the function $f(x)$ being expanded is *flat* w. r. t. $b(x)$, i. e. $f(x) \in O(b(x)^e)$ for all e
(2) demand that $f(x) = 0$ eventually

These two differ only if $f(x)$ goes to 0 faster than we can measure with our basis (e. g. $\exp(-x)$ with the basis x). Our algorithm never produces such 'expansions', so we chose the stronger option (2).

We then define *wf* to be a coinductive predicate given by these three rules. This means that *wf* holds iff there is some finite or infinite derivation tree using these three rules. This is equivalent to the Poincaré expansion definition (with the caveat about expansions of finite length), but we never show this connection in Isabelle since we do not need it. That this definition of *wf* makes sense can be seen from the following theorem:

Theorem 7.1 (Connection between *wf* and $\sim$). *If* $\mathit{wf}_{bs}(\mathcal{F})$ *for a well-formed basis bs and a trimmed multiseries* $\mathcal{F}$ *with leading monomial* $cb_1(x) \cdot \ldots \cdot b_n(x)$*, then*

$$\mathcal{F}(x) \sim cb_1(x) \cdot \ldots \cdot b_n(x)\ .$$

It remains to show the correctness of the multiseries operations we defined. As an example, the correctness theorems for addition and the multiplicative inverse have the following form, assuming a well-formed basis bs:

$$\mathrm{wf}_{bs}(\mathcal{F}) \wedge \mathrm{wf}_{bs}(\mathcal{G}) \Longrightarrow \mathrm{wf}_{bs}(\mathcal{F} \oplus \mathcal{G})$$

$$\mathrm{wf}_{bs}(\mathcal{F}) \wedge \mathrm{trimmed}(\mathcal{F}) \Longrightarrow \mathrm{wf}_{bs}(\mathcal{I}(\mathcal{F}))$$

Here, *trimmed* is a predicate that states that $\mathcal{F}$ is trimmed. All of these proofs are straightforward inductions over bs where the recursive case requires *coinduction* w. r. t. the coinductive predicate wf. As a simple example, let us consider the correctness proof of the negation operation:

Theorem 7.2 (Correctness of series negation). *If bs is a well-formed basis and* $\mathit{wf}_{bs}(\mathcal{F})$ *holds, then* $\mathit{wf}_{bs}(\ominus\mathcal{F})$.

Proof. We proceed by induction over the basis. The case for an empty basis is trivial. Let us therefore consider a basis of the shape $b \# bs$. Then applying coinduction w. r. t. the *wf* predicate gives us the following proof obligation:

$$\begin{aligned}
&\forall \mathcal{F}.\ \mathit{wf}_{b\#bs}(\mathcal{F}) \Longrightarrow \\
&\qquad \ominus\mathcal{F} = [] \ \wedge\ \big(\underset{\infty}{\forall}x.\ -\mathcal{F}(x) = 0\big) \ \vee \\
&\qquad \exists\, C\ e\ \bar{\mathcal{F}}.\ \ominus\mathcal{F} = [\![Cb(x)^e \mid \ominus\bar{\mathcal{F}}]\!] \ \wedge \\
&\qquad\qquad (\forall x.\ -\mathcal{F}(x) - C(x)b(x)^e = -\bar{\mathcal{F}}(x)) \ \wedge \\
&\qquad\qquad \mathrm{wf}_{bs}(C) \wedge \mathrm{wf}_{b\#bs}(\bar{\mathcal{F}}) \wedge \\
&\qquad\qquad (\forall e' > e.\ -\mathcal{F}(x) \in o(b(x)^{e'}) \ \wedge \\
&\qquad\qquad e > \mathrm{lead\_exp}(\ominus\bar{\mathcal{F}})
\end{aligned}$$

By unfolding one step of the corecursive definition of $\ominus$, this simplifies to the following two cases:

Case 1: $\mathcal{F} = [\![]\!]$
Then $\mathit{wf}_{b\#bs}(\mathcal{F})$ implies $\underset{\infty}{\forall}x.\ \mathcal{F}(x) = 0$ by definition and the proof obligation simplifies to $\underset{\infty}{\forall}x.\ -\mathcal{F}(x) = 0$, which is then obviously true.

Case 2: $\mathcal{F} = [\![Cb(x)^e \mid \bar{\mathcal{F}}]\!]$ for some $C, e, \bar{\mathcal{F}}$
It is clear that the values C, e, and $\bar{\mathcal{F}}$ in the existential quantifier must be instantiated with $\ominus C$, e, and $\bar{\mathcal{F}}$, respectively. After simplification, all the proof obligations follow trivially from the induction hypothesis and the definitions. □

Series negation is certainly one of the easiest operations, but the above proof still illustrates three things:

- The proof obligations, even for simple operations, can get quite big and confusing.
- The case distinctions that have to be made are obvious from the definition of the operation.
- Once the right case distinctions are made, the proof obligations become much simpler and seem very obvious.

The proofs for more involved operations are larger and have more cases, but otherwise very similar to this one. Ultimately, writing these proofs is relatively easy due to the guidance from Isabelle. It is always obvious what has to be done, even though this is somewhat obscured by the fact that Isabelle presents the proof obligations in the rather unwieldy form of disjunctions of existential quantifiers that we saw above. We also modified the definitions of the operations several times and were able to adjust the correctness proofs with little effort.

8 COMPUTING AND USING EXPANSIONS

Isabelle is written in Standard ML and exposes this interface to users to add new tools on-the-fly. While the notions of multiseries and expansions is fully formalised in the system, the procedure to *compute* them is not; it is merely ML code. This makes the procedure much easier to implement and more flexible since it lives *outside* the system, but a disadvantage is that we cannot reason about it *inside* the system – e. g. we cannot prove that it is correct or terminates. However, due to the architecture of Isabelle, a mistake in our procedure would result in a run-time error or non-termination at worst, but never an incorrect result since all reasoning performed by the procedure still passes through the Isabelle kernel.

The basic procedure is fairly simple:

(1) Convert the expression defining the function $f(x)$ that is to be expanded into an AST.
(2) Find expansions 'bottom up' to produce a theorem of the form $wf_{bs}(\mathcal{F})$ with $\mathcal{F}(x) = f(x)$. Trim *only* when necessary.
(3) Trim the resulting multiseries until the desired result can be read off.

To perform the trimming, the multiseries expressions (which are Isabelle terms) need to be 'evaluated' partially until the leading monomial can be read off. To this end, we wrote a small lazy evaluation framework for Isabelle terms that supports lazy pattern matching on terms and returns a theorem showing that the reduced term is indeed equal (w. r. t. equality in Higher-Order Logic) to the original term.

The entire procedure is then packaged into a *proof tactic* called `real_asymp` that can be applied to problems of the form

- $f(x) \xrightarrow{x \to G} F$
- $f(x) \in L_{[F]}(g(x))$ where L is any of the five Landau symbols
- eventually $f(x) \leq g(x)$ w. r. t. F
- $f(x) \sim_{[F]} g(x)$

where F and G are filters corresponding to either the full / pointed / left / right neighbourhood of a real number or ∞ / $-\infty$ / $\pm\infty$. This tactic constitutes the most important part of this work: Isabelle is a document-oriented *theorem prover*, so the typical use case is that a user will already know the limit of the function, write down the corresponding statement, and then prove it with our method. However, for convenience, we also added the diagnostic commands `real_limit` and `real_expansion` that display the limit (resp. an initial fragment of the multiseries expansion) of a given function.

Since some more advanced functions and their asymptotic behaviour are not available in Isabelle's core library but only in the external *Archive of Formal Proofs*, we also provide an interface to register new user-defined functions with the expansion procedure at a later time. This is done to achieve partial support for the Γ and erf functions.

'Oscillating' expressions like $\sin(x)$ for $x \longrightarrow \infty$, $(-1)^n$, or $\lfloor x \rfloor$ do not have a multiseries expansion, but limited support for them is provided using what we call 'asymptotic interval arithmetic': When encountering such an expression $f(x)$, we attempt to compute bound functions $l(x)$ and $u(x)$ with known multiseries expansions such that $l(x) \leq f(x) \leq u(x)$. For example, for $\sin(x)$ and $(-1)^n$ the bounds would be $[-1; 1]$; for $\lfloor x \rfloor$ they would be $[x - 1; x]$. This method is clearly not complete since it does not handle cancellations of any kind; e. g. the bounds computed for $\sin(x) - \sin(x)$ would be $[-2; 2]$. Nevertheless, this is enough to handle many interesting cases, e. g. $\sin(x)/x \xrightarrow{x \to \infty} 0$ or $\ln x - \ln\lfloor x \rfloor \in O(1/x)$.

9 LIMITATIONS

There are three limitations of the current implementation:

Zero-checking. As mentioned before, recognising whether a given constant (even an exp-log constant) is zero is difficult. The two methods that we implemented so far use Isabelle's simplifier and interval arithmetic to attack this problem. This works well for many interesting examples, and whenever this fails, the user can simply prove the corresponding fact by hand and add it to the hypotheses. Automating this further would be desirable, but would require great improvements to Isabelle's automation for arithmetic reasoning.

Worst-case performance. It is well-known that the algorithm implemented here has very poor worst-case behaviour; e. g. Richardson *et al.* [8] give the example

$$\frac{1}{1 - \frac{1}{x}} - \frac{1}{1 - \frac{1}{x}} + x^{-n} \ .$$

Here, many initial zeros need to be trimmed before arriving at the x^{-n} term. The algorithm therefore takes at least linear time in n, leading to poor performance for large n. A possible solution for this problem is given by van der Hoeven [13] in the form of *cartesian representations*, but it would be highly non-trivial to integrate this approach with our current work and we have not observed such severe performance problems in our examples.

Non-exp-log functions. The 'oscillating' functions sin, cos, and tan are only fully supported if their argument is bounded. More comprehensive approaches for sin etc. exist [10], but to our knowledge, these are not used in practice.

Functions like arctan, Γ, and erf are only partially supported: $\Gamma(x)$ and $\text{erf}(x)$ are currently only supported for $x \longrightarrow \infty$ since other cases tend to involve complicated constants that Isabelle's automation cannot handle well.

As a more fundamental problem, for any of these functions, we cannot expand expressions like $f(x + \exp(-x))$ where the larger basis element $\exp(x)$ is not present in the leading term of the argument of f but *is* present in terms of smaller order. Implementing this requires substituting a multiseries with leading exponent 0 into an asymptotic power series. Shackell [12] shows how to do this, but it is not clear to us how to formally prove that this is correct.

10 EVALUATION

We evaluated our procedure on a large number of examples, including the list given in Section 8 of Gruntz's PhD thesis [4]. According to Gruntz, this list consists of various problems that were difficult for computer algebra systems in the early 1990s. Indeed Maxima still returns an incorrect result for one of them.[2] All of the 20 exp-log examples can be proven fully automatically by our tactic within less than a second (average 0.24 s, maximum 0.65 s). Of the 17 non-exp-log examples, most lie outside the scope of our procedure due to unsupported functions (e. g. Bessel functions) or our incomplete support for Γ and erf. The 5 that are supported, however, all work in $\leq 2.5\,s$. Our list of examples can be found in the file `src/HOL/Real_Asymp/Real_Asymp_Examples.thy` of the Isabelle distribution. A short user manual is also provided.

Let us return to our introductory example from Figure 1 and compare our tactic with two CASs capable of handling additional assumptions on the parameters, Mathematica and Maple.

Maple computes the limit to be 0 in 0.8 s. However, this alone does not tell us that the approach is from the right. If we force Maple to really compute the leading term of the expansion by multiplying the function with $\ln^{1+\frac{\varepsilon}{2}} x$ and then asking for the limit, it gives up almost immediately and returns the unevaluated expression.

Mathematica computes the limit to be 0 in 21 s. If we force it to compute the leading term, it produces the correct result in 463 s, but it also imposes the unnecessary side condition $\varepsilon < 1$.

Our tactic fails on this example with an error message indicating that it could not determine the sign of $\ln b \cdot \varepsilon$. The reason for this is that the simplifier does not use the theorem $x < 0 \wedge y > 0 \Longrightarrow x \cdot y < 0$ by default. If we add this to the simplifier's rewriting rules, the tactic successfully proves the limit to be 0^+ in 0.3 s. It is even able to compute the leading term $-\frac{1}{2}\varepsilon \ln b \cdot \ln^{-1-\frac{\varepsilon}{2}} x$ in 0.4 s.

All in all, these examples show that, within its scope, our tactic works well and perhaps even handles parameters somewhat better than proprietary CASs like Mathematica and Maple. In general, however, these CASs are of course much better both in terms of performance and number of supported functions. Still, we are pleased with this result, especially considering the immense additional difficulty imposed by working inside a proof assistant where we want not only a *result*, but a *proof*.

11 RELATED WORK

The first complete algorithm for the computation of exp-log functions was given by Shackell in 1990 [11]. Gruntz built on this algorithm and implemented it in Maple [4]. Gruntz's algorithm was later also implemented in Mathematica by Richter [9]. As stated before, our work builds directly on the *multiseries* approach presented by Richardson *et al.* [8].

In our prior work on the proof of the Akra–Bazzi theorem [3], we introduced some very simple automation related to asymptotics, e. g. to automatically prove or disprove statements of the form $f(x) \in O(g(x))$ where f and g are products of powers of iterated logarithms, e. g. $x \ln x \in O(x^2 (\ln \ln x)^3)$. Apart from Isabelle/HOL, there are some other systems that have a library around limits and asymptotics (e. g. Coq, HOL Light, Mizar), but to our knowledge, none of them has any automation for proving limits or other asymptotic properties for any non-trivial class of problems.

[2] Maxima claims that $\lim_{x\to\infty} \ln\ln(x + \exp(\ln x \ln\ln x))/\ln\ln\ln(\exp(x) + x) = 0$, whereas the correct result is 1. We reported this bug in February 2018 but have not received a response.

12 CONCLUSION

We provide the first implementation of automated real asymptotics inside a proof assistant. The procedure provides:

- full support for basic arithmetic, exp, ln, roots, and $|\cdot|$
- full support for sin, cos, and tan at finite points
- 'best effort' support using interval arithmetic for oscillating functions like sin, cos, tan at infinity and $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$
- partial support for arctan, Γ, and erf
- support for parameters

All results produced by the procedure are trustworthy by construction as they pass through the Isabelle kernel, which reduces them down to definitions and basic logical inferences. On most practical examples, the procedure works quickly and fully automatically.

Acknowledgements. We would like to thank Joris van der Hoeven and Bruno Salvy for their help in understanding Multiseries and Maximilian P. L. Haslbeck and Kristina Magnussen for comments on a draft of this paper. We also thank Andrei Popescu for helping us with coinduction, and the reviewers for their comments. This work was supported by DFG grant NI 491/16-1.

REFERENCES

[1] Julian Biendarra, Jasmin Christian Blanchette, Aymeric Bouzy, Martin Desharnais, Mathias Fleury, Johannes Hölzl, Ondřej Kunčar, Andreas Lochbihler, Fabian Meier, Lorenz Panny, Andrei Popescu, Christian Sternagel, René Thiemann, and Dmitriy Traytel. 2017. Foundational (Co)datatypes and (Co)recursion for Higher-Order Logic. In *Frontiers of Combining Systems*, Clare Dixon and Marcelo Finger (Eds.). Springer International Publishing, Cham, 3–21.

[2] Nicolas Bourbaki. 1971. *Topologie générale.*

[3] Manuel Eberl. 2017. Proving Divide and Conquer Complexities in Isabelle/HOL. *Journal of Automated Reasoning* 58, 4 (01 Apr 2017), 483–508. https://doi.org/10.1007/s10817-016-9378-0

[4] Dominik Gruntz. 1996. *On Computing Limits in a Symbolic Manipulation System.* Ph.D. Dissertation. ETH Zürich.

[5] Johannes Hölzl. 2009. Proving Inequalities over Reals with Computation in Isabelle/HOL. In *Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS'09)*, Gabriel Dos Reis and Laurent Théry (Eds.). Munich, 38–45.

[6] Johannes Hölzl, Fabian Immler, and Brian Huffman. 2013. Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In *Proceedings of the 4th International Conference on Interactive Theorem Proving (ITP'13).* Springer-Verlag, Berlin, Heidelberg, 279–294. https://doi.org/10.1007/978-3-642-39634-2_21

[7] Daniel Richardson. 1995. A Simplified Method of Recognizing Zero Among Elementary Constants. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95).* ACM, New York, NY, USA, 104–109. https://doi.org/10.1145/220346.220360

[8] Daniel Richardson, Bruno Salvy, John Shackell, and Joris Van der Hoeven. 1996. Asymptotic Expansions of exp-log Functions. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC '96).* ACM, New York, NY, USA, 309–313. https://doi.org/10.1145/236869.237089

[9] Udo Richter. 2005. *Automatische Berechnung von Grenzwerten und Implementierung in Mathematica.* diploma thesis. Universität Kassel.

[10] Bruno Salvy and John Shackell. 2010. Measured limits and multiseries. *Journal of the London Mathematical Society* 82, 3 (2010), 747–762. https://doi.org/10.1112/jlms/jdq057

[11] John Shackell. 1990. Growth estimates for exp–log functions. *Journal of Symbolic Computation* 10, 6 (1990), 611–632. https://doi.org/10.1016/S0747-7171(08)80161-7

[12] John R. Shackell. 2004. *Symbolic Asymptotics.* Algorithms and Computation in Mathematics, Vol. 12. Springer Berlin–Heidelberg.

[13] Joris van der Hoeven. 1997. *Automatic asymptotics.* Ph.D. Dissertation. École polytechnique, Palaiseau, France.

[14] Joris van der Hoeven. 2006. *Transseries and real differential algebra.* Lecture Notes in Mathematics, Vol. 1888. Springer-Verlag.

B Divide-and-Conquer Recurrences

This chapter was originally published as an article in a peer-reviewed journal:

M. Eberl.
Proving Divide and Conquer Complexities in Isabelle/HOL.
Journal of Automated Reasoning, 58(4):483–508, Apr 2017, Springer Netherlands. © 2016
DOI: `10.1007/s10817-016-9378-0`.

I am the sole author of this article, thus all contributions are mine.

Synopsis: This work gives the first formal proof of the *Akra–Bazzi Theorem*, a 'cooking book' method to determine the asymptotic growth of solutions to the divide-and-conquer recurrences that are often encountered in the analysis of divide-and-conquer algorithms. As a simple corollary of this, a generalised version of the well-known *Master Theorem* for divide-and-conquer recurrences is also obtained, including a proof method to facilitate applying it to concrete examples with minimal manual work. The usability of this is demonstrated on several textbook examples such as *merge sort* and *Karatsuba multiplication*.

J Autom Reasoning (2017) 58:483–508
DOI 10.1007/s10817-016-9378-0

Proving Divide and Conquer Complexities in Isabelle/HOL

Manuel Eberl[1]

Received: 23 July 2015 / Accepted: 3 June 2016 / Published online: 13 June 2016

Abstract The Akra–Bazzi method (Akra and Bazzi in Comput Optim Appl 10(2):195–210, 1998. doi:10.1023/A:1018373005182), a generalisation of the well-known Master Theorem, is a useful tool for analysing the complexity of Divide and Conquer algorithms. This work describes a formalisation of the Akra–Bazzi method (as generalised by Leighton in Notes on better Master theorems for divide-and-conquer recurrences, 1996. http://courses.csail.mit.edu/6.046/spring04/handouts/akrabazzi.pdf) in the interactive theorem prover Isabelle/HOL and the derivation of a generalised version of the Master Theorem from it. We also provide some automated proof methods that facilitate the application of this Master Theorem and allow mostly automatic verification of Θ-bounds for these Divide and Conquer recurrences. To our knowledge, this is the first formalisation of theorems for the analysis of such recurrences.

Keywords Isabelle/HOL · Master Theorem · Akra–Bazzi · Divide and Conquer algorithms · Recurrences · Complexity · Landau symbols

1 Introduction

The *Master Theorem* is the textbook method taught in undergraduate algorithms lectures for analysing the asymptotic run-time complexity of many Divide and Conquer algorithms. The general form of the admissible algorithms is several non-recursive base cases and one recursive case, in which a problem of size n is reduced to a fixed number a of subproblems of size n/b, which are then solved recursively and their solutions combined to a solution for the original problem. One simple example is *Merge Sort*: lists of length ≤ 1 are always trivially sorted and thus returned unchanged (the base cases); lists of size ≥ 2 are split into half, each

This work was supported by DFG RTG 1480 (PUMA).

✉ Manuel Eberl
eberlm@in.tum.de

1 Fakultät für Informatik, Technische Universität München, Garching, Germany

half is sorted recursively, and the two sorted halves are then combined into a single sorted list. The recurrence relation of this algorithm's run-time function T is

$$T(n) = 2T\left(\tfrac{1}{2}n\right) + n$$

and the classical Master Theorem then states $T \in \Theta(n \ln n)$. One problem with this is that, strictly speaking, the recurrence relation for T is actually something like

$$T(n) = T\left(\left\lfloor \tfrac{1}{2}n \right\rfloor\right) + T\left(\left\lceil \tfrac{1}{2}n \right\rceil\right) + n$$

since one certainly cannot split a list of length 3 into two lists of length 1.5. Intuitively, one may think that the rounding does not change the asymptotic behaviour of the function, seeing as the rounding operations are asymptotically small disturbances. This is, in fact, the case, but proving it is not entirely trivial and is seldom done rigorously in textbooks or lectures, especially the case when both floors and ceilings are used at the same time, as in the example above.[1]

In order to establish a basis for the verified run-time analysis of such algorithms, we wanted to formalise the Master Theorem in the theorem prover Isabelle/HOL. This must, of course, include rigorous handling of rounding operations. We therefore chose not to base our formal proof on any of the literature proofs for the Master Theorem, but to instead prove a generalisation known as the *Akra–Bazzi method* [1], and derive the Master Theorem as a corollary. As a pleasant side effect, this version of the Master Theorem supports much more complex recursion patterns than the classical Master Theorem from the literature.

To make the application of the Master Theorem in the theorem prover almost as simple as on paper, we provided some proof automation machinery that facilitates the definition of functions from Akra–Bazzi-type Divide and Conquer recurrences and allows applying the Master Theorem to them in a mostly automatic way.

To provide some motivation, consider the following two recurrences, which are related to a deterministic selection algorithm and so-called Ham-Sandwich trees. They are far outside the scope of the classical Master Theorem, but their complexities can easily be found and formally proven in Isabelle/HOL with the generalised Master Theorem we formalised:

$$f_1(n) = f_1\left(\left\lfloor \tfrac{n}{5} \right\rfloor\right) + f_1\left(\left\lfloor \tfrac{7n}{10} \right\rfloor + 6\right) + \tfrac{12n}{5} \qquad \text{Result: } f_1 \in \Theta(n)$$

$$f_2(n) = f_2\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) + f_2\left(\left\lfloor \tfrac{n}{4} \right\rfloor\right) + 1 \qquad \text{Result: } f_2 \in \Theta(n^{\log_2 \varphi}) \text{ where } \varphi = \tfrac{1+\sqrt{5}}{2}$$

Outline In Sect. 2, we will list some related work on the type of recurrences that we focus on in this work. Section 3 then gives some important background information, namely about the notation used in this work and the notions of integration and Landau symbols that are used in the formal proof.

Section 4 contains a formal description of the types of recurrences for which our results—the Akra–Bazzi theorem and the generalised Master Theorem—hold, and Sects. 5 and 6 state these results. The formal Isabelle/HOL proof of the results is explained in Sect. 7. Section 8 gives a list of the proof automation we developed for the Master Theorem as well as a few examples of its application.

Finally, Sect. 9 compares the scope of our version of the Akra–Bazzi theorem to Leighton's and our Master Theorem to the 'textbook' version of the Master Theorem.

[1] In *Introduction to Algorithms* [7], for example, only the 'floor' case is proven and the 'ceiling' case is stated to be analogous. The question of what happens when both floors and ceilings are used is not addressed.

2 Related Work

The original paper by Akra and Bazzi [1] that introduced the Akra–Bazzi method uses a so-called *order transform* to reduce the problem to a two-dimensional problem. Their version of the method requires very strong assumptions on the parameters of the problem: the recursive definition must be of the following form:

$$f(x) = g(x) + \sum_{i=1}^{k} a_i \cdot f(\lfloor \tfrac{x}{b_i} \rfloor) \qquad \left(b_i \in \mathbb{N}_{\geq 2},\ g \text{ non-decreasing}\right)$$

In particular, recursive calls like $f(\lceil \frac{x}{2} \rceil)$, $f(\lfloor \frac{2}{3}x \rfloor)$, or $f(\lfloor \frac{1}{3}x \rfloor + 1)$ are not allowed.

Leighton [14] gives a vastly generalised version of the theorem in which the above restrictions on g and the recursive call are weakened greatly; in particular, his version allows small deviations from the linear recursive call, which, among other things, includes rounding and adding constants. Furthermore, his approach is much more direct; he gives a simple inductive proof that we deemed much more amenable to formal verification than the original proof by Akra and Bazzi. The statement of the theorem we proved and its formal proof in Isabelle/HOL are therefore modelled very closely after his; the only differences in the theorem statement are that some assumptions in our version have been weakened slightly (e. g. allowing negative values on some initial segment of the domain).

Based on these two works, Bazzi and Mitter [5] give a version of the Akra–Bazzi theorem for probabilistic recurrences, where both the factors b_i and the non-linear deviations in the recursive calls are random variables with some restrictions. The theorem then determines the asymptotic growth of the expectation of the function thus defined in a way that is very similar to the theorem given by Leighton.

Drmota and Szpankowski [8] analyse recurrences of the form

$$f(x) = g(x) + \sum_{i=1}^{m} a_i \cdot f\left(\left\lfloor b_j \cdot x + O\left(x^{1-\varepsilon}\right)\right\rfloor\right) + \sum_{i=1}^{m} \bar{a}_i \cdot f\left(\left\lceil \bar{b}_j \cdot x + O\left(x^{1-\varepsilon}\right)\right\rceil\right)$$

with the additional restriction that the $b_j \cdot x + O(x^{1-\varepsilon})$ are increasing. The class of functions that they consider is therefore smaller than Leighton's and ours. However, while Leighton and we are only interested in finding a Θ-bound for f, Drmota and Szpankowski obtain very precise approximations for f, such as $f(x) = C_2 \log n + C_2' + o(1)$ for explicitly computable constants C_2, C_2'. They also describe the oscillations that arise in f due to the rounding in the recurrence and which are not present in the corresponding continuous recurrences. For the Θ-analysis, these oscillations are irrelevant, since they are asymptotically small.

3 Preliminaries

3.1 Syntactical Note

We take some liberties when presenting expressions or theorems from Isabelle/HOL here to increase readability. In particular: type coercions between real numbers and natural numbers are always omitted; schematic variables, which are implicitly universally quantified in Isabelle, are printed with an explicit $\forall$ for the sake of clarity; Isabelle-specific syntax, such as $\{0\,..{<}1\}$ is replaced with the standard notation $[0;\,1)$; lists are sometimes implicitly used as sets or as indexed sequences (e. g. as_i for the i-th element of as, starting from 1).

Table 1 Definitions of the five Landau symbols

$f \in O(g)$	$\longleftrightarrow$	$\exists c > 0.\ \exists x_0.\ \forall x \geq x_0.\ \lvert f(x)\rvert \leq c \cdot \lvert g(x)\rvert$
$f \in o(g)$	$\longleftrightarrow$	$\forall c > 0.\ \exists x_0.\ \forall x \geq x_0.\ \lvert f(x)\rvert \leq c \cdot \lvert g(x)\rvert$
$f \in \Omega(g)$	$\longleftrightarrow$	$\exists c > 0.\ \exists x_0.\ \forall x \geq x_0.\ \lvert f(x)\rvert \geq c \cdot \lvert g(x)\rvert$
$f \in \omega(g)$	$\longleftrightarrow$	$\forall c > 0.\ \exists x_0.\ \forall x \geq x_0.\ \lvert f(x)\rvert \geq c \cdot \lvert g(x)\rvert$
$f \in \Theta(g)$	$\longleftrightarrow$	$f \in O(g) \wedge f \in \Omega(g)$

A function taking some value x and returning some t that may depend on x will be written as $\lambda x.\ t$, following Lambda calculus syntax as opposed to the traditional mathematical syntax $x \mapsto t$. We will occasionally omit the 'λ' when it is clear from the context that we mean a function and what the function variable is, particularly in Landau symbols (e. g. $x \in O(x^2)$ instead of $(\lambda x.\ x) \in O(\lambda x.\ x^2)$).

3.2 Landau Symbols

3.2.1 Definition

Before stating the Akra–Bazzi theorem, we shall give the precise definition of the Landau symbols that were used in the Isabelle formalisation [10]. Since there was no suitable formalisation of asymptotic growth in Isabelle/HOL, we created a library of Landau symbols specifically for the formalisation of the Akra–Bazzi method, but with other use cases in mind as well. The definitions we chose differ slightly from those given in *Introduction to Algorithms* by Cormen et al. [7]: for $f \in O(g)$ to hold, they require f and g to be positive for sufficiently large inputs, whereas we do not. According to our definitions,

$$f \in O(g) \longleftrightarrow -f \in O(g) \longleftrightarrow f \in O(-g) \longleftrightarrow -f \in O(-g).$$

This choice was made because, in our experience, formal reasoning with Landau symbols—especially automatic reasoning—becomes easier this way.

Table 1 shows the definitions of our Landau symbols. In Isabelle, Landau symbols are defined for functions from any (not necessarily linearly) ordered set to any linearly-ordered field. The restriction to fields was made due to the fact that the existence of multiplicative inverses makes Landau symbols more 'well-behaved', and one may even argue that Landau symbols for functions into e. g. the natural numbers do not make much sense—one will probably always want to view such functions as functions into the reals.

There is one prior formalisation of Landau symbols in Isabelle by Avigad et al. [2]. It is related to their proof of the Prime Number theorem. They only defined the symbol 'O', and they did so in the following fashion:

$$O(f) = \{h \mid \exists c.\ \forall x.\ \lvert h(x)\rvert \leq c \cdot \lvert f(x)\rvert\}$$

This definition differs from the commonly used one in so far as this one requires the inequality to hold on all inputs, not just for sufficiently large inputs. If the inputs are natural numbers, the two are almost equivalent, but in the context of the Akra–Bazzi theorem, we also use Landau symbols for functions of type $\mathbb{R} \to \mathbb{R}$ and then these two definitions are quite different. We therefore deemed it necessary to create our own library of Landau symbols.

3.2.2 Decision Procedure

During the process of proving the generalised Master Theorem, and particularly in its applications, we encountered a great number of proof obligations like $x \in O(x^3)$, $x^2 \in o(x^2 \ln x)$, $\ln x \in \omega(\ln \ln x)$, etc. These problems all have the following in common:

- They are of the form $f \in L(g)$, where L is a Landau symbol and $f, g : \mathbb{R} \to \mathbb{R}$ are products of ‘elementary building blocks’ like the identity function and iterated logarithms.
- The building blocks can be ordered linearly w. r. t. their growth (e.g. x, $\ln x$, $\ln \ln x$,...) in such a way that the growth rate of any positive power of a function in the sequence eclipses that of any power of the subsequent one (e.g. $x^p \in \omega((\ln x)^q)$ for any $p, q \in \mathbb{R}$ with $p > 0$).
- These problems are typically very tedious to prove formally.
- Most mathematicians would dismiss them as trivial and not even bother proving them by hand in a pen-and-paper proof.

The obvious course of action was therefore to develop automation machinery to discharge these proof obligations automatically. In the following, we will sketch the decision procedure we developed for this problem without going into too much detail.

Definition 1 (*Families of functions*) We call $\mathcal{F} \subseteq \mathbb{R}^{\mathbb{R}}$ a *family* of functions if

- $\mathcal{F}$ is closed under multiplication and multiplicative inverse
- each function in $\mathcal{F}$ is positive for all sufficiently large inputs
- $\mathcal{F}$ is linearly ordered in the sense that for any $f, \bar{f} \in \mathcal{F}$, at least one of $f \in o(\bar{f})$, $f \in \omega(\bar{f})$, and $f \in \Theta(\bar{f})$ holds

Examples for such families are $\{\lambda x.\ x^p \mid p \in \mathbb{R}\}$ or $\{\lambda x.\ a^x \mid a \in \mathbb{R}_{>0}\}$.

Definition 2 (*Dominating families*) We say that a family $\mathcal{F}$ *dominates* a family $\mathcal{G}$ if

- there exists an $f \in \mathcal{F}$ such that $g \in o(f)$ for all $g \in \mathcal{G}$
- $f(x) \in o(\bar{f}(x))$ implies $f(x) \cdot g(x) \in o(\bar{f}(x) \cdot \bar{g}(x))$ for any $f, \bar{f} \in \mathcal{F}$ and $g, \bar{g} \in \mathcal{G}$

If $\mathcal{F}$ dominates $\mathcal{G}$, we immediately have for all $f, \bar{f} \in \mathcal{F}$ and $g, \bar{g} \in \mathcal{G}$:

$$f(x) \cdot g(x) \in o(\bar{f}(x) \cdot \bar{g}(x)) \longleftrightarrow f \in o(\bar{f}) \vee (f \in \Theta(\bar{f}) \wedge g \in o(\bar{g}))$$
$$f(x) \cdot g(x) \in O(\bar{f}(x) \cdot \bar{g}(x)) \longleftrightarrow f \in o(\bar{f}) \vee (f \in \Theta(\bar{f}) \wedge g \in O(\bar{g}))$$
$$f(x) \cdot g(x) \in \Theta(\bar{f}(x) \cdot \bar{g}(x)) \longleftrightarrow f \in \Theta(\bar{f}) \wedge \bar{g} \in \Theta(\bar{g})$$

In other words: o, O, and Θ on $\mathcal{F} \cdot \mathcal{G}$ behave analogously to $<$, $\leq$, and $=$ on pairs with lexicographic ordering.

Furthermore, it is obvious that if $\mathcal{F}$ dominates $\mathcal{G}$ and $\mathcal{G}$ dominates $\mathcal{H}$, then $\mathcal{F}$ dominates $\mathcal{G} \cdot \mathcal{H}$. This notion of transitivity implies that we can lift the above result on pairs to sequences where each family dominates the next. We can thus reduce any statement of the form

$$f_1(x) \ldots f_n(x) \in L(\bar{f}_1(x) \ldots \bar{f}_n(x)) \qquad (\text{with } L \in \{o, O, \Theta\} \text{ and } f_i, \bar{f}_i \in \mathcal{F}_i)$$

to a statement involving only Boolean connectives and expressions of the form $f_i \in l(\bar{f}_i)$ for $l \in \{o, O, \Theta\}$.

Of course, this means that if the $\mathcal{F}_i$ are chosen such that we can decide $f_i \in l(\bar{f}_i)$, we can also decide $f \in L(\bar{f})$. In our decision procedure, the admissible function families are powers of iterated logarithms; i. e. for each fixed $k \in \mathbb{N}$, the functions of the form

$$\lambda x.\ (\underbrace{\ln\ \ldots\ \ln}_{k \text{ times}}\ x)^p \qquad (\text{for some } p \in \mathbb{R})$$

form one family. Deciding $f_k \in l(\bar{f_k})$ for two functions in such a family can then be done easily by comparing the exponents.

Our decision procedure therefore simply analyses a goal like $x \in o(x \ln x)$ and rewrites it to $x^1 (\ln x)^0 \in o(x^1 (\ln x)^1)$. By the above result, this holds iff $1 < 1 \vee (1 = 1 \wedge 0 < 1)$, which Isabelle's simplifier can easily prove automatically. We integrated this decision procedure into Isabelle's simplifier, so proof obligations of this form will automatically be rewritten to necessary and sufficient conditions containing only Boolean connectives and comparisons on the exponents. Additionally, if these exponents are numeric constants, the conditions are then proven (or disproven) automatically by the simplifier. This made many of our proofs and the application of our main results much easier.

In addition to this decision procedure, we also have simplifier setup that can:

- Simplify terms like $L(f + g)$ to $L(g)$ if $f \in o(g)$ or to $L(f)$ if $g \in o(f)$
- Cancel common factors like h from $f \cdot h \in L(g \cdot h)$ if $h(x)$ is non-zero for large enough x
- Perform simplifications on functions under a Landau symbol that are valid for sufficiently large values, e. g. $\ln(2x) = \ln 2 + \ln x$. (This is not valid in Isabelle/HOL for $x \leq 0$)

We initially formalised the proof of the asymptotic inequalities described in Sect. 7.2 in an elementary way. The resulting proofs were complex and virtually unreadable. After the introduction of Landau symbols and with heavy use of the automation we just described, we were able to more than halve the length of these proofs and make them significantly more readable.

3.3 Integration

Since the statement of the Akra–Bazzi theorem contains an integral, we need to decide on a definition of integration and a formalisation thereof. Isabelle contains a number of different integrals, most notably the *Henstock–Kurzweil integral* (also known as the *Gauge integral*) on functions from Euclidean spaces to normed real vector spaces and the *Bochner integral* (an extension of the Lebesgue integral) on functions from a measure space to the real numbers [3].

In the proof of the Akra–Bazzi theorem, we noticed that the only properties of integration that we actually needed are the following:

Integral of constant functions If $a \leq b$ and $c \in \mathbb{R}_{\geq 0}$, the constant function $\lambda x.\ c$ is integrable on $[a; b]$ and

$$\int_a^b c \, \mathrm{d}x = (b - a) \cdot c.$$

Monotonicity If f and g are integrable on $[a; b]$ and $f(x) \leq g(x)$ for all $x \in [a; b]$, then

$$\int_a^b f(x) \, \mathrm{d}x \leq \int_a^b g(x) \, \mathrm{d}x.$$

Integrability on sub-intervals If f is integrable on $[a; d]$ and $a \leq b \leq c \leq d$, then f is also integrable on $[b; c]$.

Splitting if f is integrable on $[a; c]$ and $a \leq b \leq c$, then

$$\int_a^b f(x) \, \mathrm{d}x + \int_b^c f(x) \, \mathrm{d}x = \int_a^c f(x) \, \mathrm{d}x.$$

We therefore proved the Akra–Bazzi theorem generically w. r. t. the integral definition: the theorem can be instantiated with any concept of integration and integrability that fulfils the above four properties. We call such an integral *admissible*.

The ‘standard’ integrals like the Riemann, Lebesgue, Bochner, and Henstock–Kurzweil integrals all fulfil these properties and are therefore admissible, as is the non-negative Lebesgue integral that ignores the negative part of the integrand. Notably, all of these are generalisations of the Riemann integral on non-negative functions. The natural question is then: Are there any admissible ‘integrals’ that are not integrals in the usual sense, i. e. not merely generalisations of the Riemann integral?

The answer to this is not immediately obvious, but it turns out that any admissible integral $\mathcal{I}$ must coincide with the Riemann integral on any function f that is both piecewise continuous and $\mathcal{I}$-integrable, but it can differ from the Riemann integral and its generalisations on non-piecewise-continuous functions. Since non-piecewise-continuous functions should rarely arise in the context of the Akra–Bazzi theorem, we shall not explore the issue further here; a more detailed explanation and a proof can be found in the “Appendix”.

4 General Setting

Let us now set up the context in which the remainder of this work will be set: For our version of the Akra–Bazzi method, we shall consider a recursively-defined function $f : \mathbb{N} \to \mathbb{R}$ with the following properties:

$$f(x) \geq 0 \quad \text{for all } x \in [x_0; x_1)$$

$$f(x) = g(x) + \sum_{i=1}^{k} a_i \cdot f\,(b_i \cdot x + h_i(x)) \quad \text{for all } x \geq x_1$$

for a natural number $k \in \mathbb{N} \setminus \{0\}$, a function $g : \mathbb{N} \to \mathbb{R}$, natural numbers $x_0, x_1 \in \mathbb{N}$, real coefficients $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$, functions $h_i : \mathbb{N} \to \mathbb{R}$ such that:

- $g(x) \geq 0$ for all $x \geq x_1$
- $a_i \geq 0$ for all $i \in [1; k]$ and $a_i > 0$ for at least one $i \in [1; k]$
- $b_i \in (0; 1)$ for all $i \in [1; k]$
- for every $i \in [1; k]$, there exists an $\varepsilon > 0$ such that $h_i \in O(x/\ln^{1+\varepsilon} x)$
- $b_i x + h_i(x) \in \mathbb{N}$ and $x_0 \leq b_i \cdot x + h_i(x) < x$ for all $i \in [1; k]$ and all $x \geq x_1$ (well-definedness of f)

We will now explain the meaning of these variables.

The recursion structure The parameters x_0, x_1, k, a_i, b_i, and h_i characterise the recursion structure of the function f. To understand the role of the different parameters, it is useful to look at them in the case when f describes the cost of a Divide & Conquer algorithm: the values between x_0 and x_1 are the costs of the base cases; the cost of the recursive case is defined recursively as the sum of the costs of the recursive calls and the costs of combining the results of the calls. Each triple (a_i, b_i, h_i) corresponds to a_i recursive calls of the form $b_i \cdot x + h_i(x)$; the costs of combining the results are represented by the function g.

Variation terms The h_i represent asymptotically small variation terms in the recursive call, allowing some deviation from the linear term $b_i \cdot x$. This is not merely a nice gimmick—it is actually necessary to have something like this, since, due to the discreteness of the natural numbers, a purely linear term in the recursive call is impossible. This approach covers rounding and other deviations in a uniform way, as opposed to making ad-hoc arguments why certain kinds of rounding do not change the result.

For example, the terms $f(\lfloor \frac{x}{2} \rfloor)$, $f(\lceil \frac{x}{2} \rceil)$, and $f(\lceil \frac{x}{2} \rceil + 42)$ would be admissible, as they can be expressed as $f(b \cdot x + h(x))$ for some $b \in (0; 1)$ and some $h : \mathbb{N} \to \mathbb{R}$ where $h \in O(1)$. However, much larger deviations, such as $f(\lfloor \frac{1}{2}n - \sqrt{n} \rfloor)$, are also allowed.

Of course, enough base cases must be provided (i.e. x_0 and x_1 must be chosen large enough and far enough apart) to fulfil the well-definedness conditions; a function 'definition' like $f(x) = f(\lceil \frac{3}{4}x \rceil) + 1$ for $x_1 = 3$ cannot be allowed since $f(3) = f(\lceil \frac{9}{4} \rceil) + 1 = f(3) + 1$ is contradictory.

Leighton [14] mentions that the condition that the h_i be in $O(x/\ln^{1+\varepsilon} x)$ for some $\varepsilon > 0$ is tight in some sense, since the recurrence $f(x) = 2f(x/2 + x/\ln x)$ has the asymptotic growth $x \ln^{\Theta(1)} x$, whereas the recurrence $f(x) = 2f(x/2)$ (i.e. without the variation term) has the growth $\Theta(x)$.

5 The Akra–Bazzi Method

Having established the necessary context, we will now present the main theorems of the Akra–Bazzi method for the function f. To do this, we first need to define the characteristic number of an Akra–Bazzi recurrence: The contribution of the recursion structure (without the 'recombination costs' g) to the asymptotic growth can be summarised as a single real number, which we call p. This number is defined implicitly as

$$\sum_{i=1}^{k} a_i \cdot b_i^p = 1$$

If the a_i are not all zero (which we assumed), this equation defines p uniquely. To show this, we consider the function

$$t : \mathbb{R} \to \mathbb{R}_{>0},\ x \mapsto \sum_{i=1}^{k} a_i \cdot b_i^x .$$

This function is continuous and $t(x) \xrightarrow{-\infty} \infty$ and $t(x) \xrightarrow{\infty} 0$. Therefore, by the intermediate value theorem, some p such that $\sum_{i=1}^{k} a_i \cdot b_i^p = 1$ always exists, and since t is also strictly decreasing, this p is unique.

We can now state the three variants of the Akra–Bazzi theorem, which give Ω, O, and Θ bounds on the growth of f:

Theorem 1 (Akra–Bazzi theorem, Ω version) *Fix $\bar{g} : \mathbb{R} \to \mathbb{R}$ with $g \in \Omega(\bar{g})$, i.e. $\bar{g}$ is an asymptotic lower bound for g. Assume that:*

- *$f(x) > 0$ for all sufficiently large x*
- *$\bar{g}(x) \geq 0$ for all sufficiently large x*
- *there exist real constants $c > 0$ and C with $\forall i \in [1; k].\ C < b_i$ such that for all sufficiently large x: $\forall u \in [C \cdot x; x].\ \bar{g}(u) \leq c\bar{g}(x)$*
- *$\bar{g}$ is bounded above on every real interval $[a; b]$ with $a \geq a_0$ for some a_0*
- *$\bar{g}(x)/x^{p+1}$ is integrable on any interval $[a; b]$ with $a \geq a_0$ for some a_0*

Then

$$f \in \Omega\left(x^p \left(1 + \int_t^x \frac{\bar{g}(u)}{u^{p+1}} \mathrm{d}u\right)\right)$$

for any sufficiently large t.

Theorem 2 (Akra–Bazzi theorem, O version) *Fix $\bar{g} : \mathbb{R} \to \mathbb{R}$ with $g \in O(\bar{g})$, i.e. $\bar{g}$ is an asymptotic upper bound for g. Assume that:*

- *$\bar{g}(x) \geq 0$ for all sufficiently large x*
- *there exist real constants $c > 0$ and C with $\forall i \in [1; k]. \, C < b_i$ such that for all sufficiently large x: $\forall u \in [C \cdot x; x]. \, \bar{g}(u) \geq c\bar{g}(x)$*
- *$\bar{g}(x)/x^{p+1}$ is integrable on any real interval $[a; b]$ with $a \geq a_0$ for sufficiently large a_0*

Then

$$f \in O\left(x^p \left(1 + \int_t^x \frac{\bar{g}(u)}{u^{p+1}} \mathrm{d}u\right)\right)$$

for any sufficiently large t.

Combining these two results yields:

Theorem 3 (Akra–Bazzi theorem, Θ version) *Fix $\bar{g} : \mathbb{R} \to \mathbb{R}$ with $g \in \Theta(\bar{g})$, i.e. $\bar{g}$ has the same asymptotic growth as g. Assume that:*

- *$f(x) > 0$ for all sufficiently large x*
- *$\bar{g}(x) \geq 0$ for all sufficiently large x*
- *there exist real constants $c_1, c_2 > 0$ and C with $\forall i \in [1; k]. \, C < b_i$ such that for all sufficiently large x: $\forall u \in [Cx; x]. \, c_1\bar{g}(x) \leq \bar{g}(u) \leq c_2\bar{g}(x)$*
- *$\bar{g}$ is bounded above on every real interval $[a; b]$ with $a \geq a_0$ for some a_0*
- *$\bar{g}(x)/x^{p+1}$ is integrable on any real interval $[a; b]$ with $a \geq a_0$ for some a_0*

Then

$$f \in \Theta\left(x^p \left(1 + \int_t^x \frac{\bar{g}(u)}{u^{p+1}} \mathrm{d}u\right)\right)$$

for any sufficiently large t.

The restrictions on $\bar{g}$ are somewhat technical, especially the ones of the form $\exists c_1 > 0. \, \forall u \in [Cx; x]. \, c_1\bar{g}(x) \leq \bar{g}(u)$. Leighton [14] calls these the *polynomial-growth conditions*[2] and also asserts that if $|\bar{g}'|$ is upper-bounded by a polynomial, the conditions always hold. This is incorrect, since e. g. $g(x) = 1 + \sin(x)$ is non-negative and the absolute of its derivative is upper-bounded by 1, but it does not fulfil either of the two polynomial-growth conditions.

Nevertheless, the most interesting cases are those where $\bar{g}(x)$ is of the form $x^r \ln^s x$, and as Leighton also remarks, these functions always satisfy the polynomial-growth conditions. Restricting $\bar{g}$ to this form, which we shall do in the next section, will lead us to a specialisation of the Akra–Bazzi method that is very close to the well-known Master Theorem.

Let us now analyse the conclusion of the last Akra–Bazzi theorem more closely in an informal way: expanding the product inside the Θ yields

$$f \in \Theta\left(x^p\right) + \Theta\left(x^p \int_t^x \frac{\bar{g}(u)}{u^{p+1}} \mathrm{d}u\right) .$$

Clearly, the x^p in the left summand is independent from $\bar{g}$ and would still be present even for $\bar{g} = 0$. The $\Theta(x^p)$ can therefore be seen as the inherent cost of the recursion itself, which depends only on p, which in turn is determined uniquely by the a_i and b_i. The term with the

[2] His conditions are slightly more restrictive; among other things, he requires them to hold for all $x \geq 1$.

integral on the right, on the other hand, also depends on the recombination costs $\bar{g}(x)$, and it is big whenever $\bar{g}(x)$ is big.

It is also clear that the values of the base cases are completely irrelevant (as long as they are non-negative).

6 The Master Theorem

If we look at a restricted class of functions $\bar{g}$, we can make the last two observations of the previous section a bit more precise: If $\bar{g}(x)$ is of the form x^q for some $q \in \mathbb{R}$, the conditions on $\bar{g}$ (non-negativity, polynomial growth, boundedness, integrability) are all satisfied. If the other conditions for the Akra–Bazzi theorem are satisfied, we have:

$$f \in \Theta\left(x^p\right) + \Theta\left(x^p \int_t^x u^{q-p-1}\mathrm{d}u\right) = \begin{cases} \Theta\left(x^p\right) + \Theta\left(x^q\right) = \Theta\left(x^p\right) & \text{for } q < p \\ \Theta\left(x^p\right) + \Theta\left(x^p \ln x\right) = \Theta\left(x^p \ln x\right) & \text{for } q = p \\ \Theta\left(x^p\right) + \Theta\left(x^q\right) = \Theta\left(x^q\right) & \text{for } q > p \end{cases}$$

The three cases differ in how high the inherent costs of the recursion are compared to the recombination costs. In the first case, the recombination costs are smaller than the recursion costs, which means that most of the work is done at the bottom of the recursion tree, since there are many leaves, but recombining them is cheap (‘bottom-heavy recursion’). In the third case, the recombination costs dominate and most of the work will be done recombining the results near the top of the recursion tree (‘top-heavy recursion’). In the second case, the recursion costs and the recombination costs have a similar rate of growth (‘balanced recursion’). This case can be generalised further by considering $\bar{g}(x) = x^p \ln^q x$.

This leads to our generalised Master Theorem:

Corollary 1 (Master Theorem)

Bottom-heavy recursion. *If $g \in O(x^q)$ for some $q < p$, then $f \in O(x^p)$. If, additionally, $f(x)$ is positive for all sufficiently large x, we even have $f \in \Theta(x^p)$.*[3]

Balanced recursion.

If $g \in \Theta(x^p \ln^q x)$ for some q, then

$$f \in \begin{cases} \Theta(x^p) & \text{if } q < -1 \\ \Theta(x^p \ln \ln x) & \text{if } q = -1 \\ \Theta(x^p \ln^{q+1} x) & \text{if } q > -1 \end{cases}$$

Top-heavy recursion. *If $g \in \Theta(x^q)$ for some $q > p$, then $f \in \Theta(x^q) = \Theta(g)$.*

7 Proving the Akra–Bazzi Theorem and the Master Theorem

We shall now describe the proof of our version of the Akra–Bazzi theorem. Some parts of the proof are very technical; in these parts of the proof, we shall attempt to provide the reader with a good high-level understanding of what must be proven and how we proved it without mentioning too many details. For more details, we refer the reader to the formal proof

[3] Note that due to the other constraints on f and g, the condition that $f(x)$ is positive for all sufficiently large x must hold if either $g(x)$ is positive for all sufficiently large x or $f(x)$ is positive in all the base cases, i.e. for $x \in [x_0; x_1)$.

locale akra_bazzi_function =
fixes $x_0\ x_1\ k$ :: nat **and** $as\ bs$ :: real list **and** ts :: (nat $\Rightarrow$ nat) list **and**
f :: nat $\Rightarrow$ real **and** g :: nat $\Rightarrow$ real
assumes $k \neq 0$ **and** $\text{length}(as) = k$ **and** $\text{length}(bs) = k$ **and** $\text{length}(ts) = k$
and $\forall a \in as.\ a \geq 0$ **and** $\forall b \in bs.\ b \in (0;1)$ **and** $\exists i \in [1;k].\ as_i > 0$
and $\forall i \in [1;k].\ \text{akra\_bazzi\_term}(x_0, x_1, bs_i, ts_i)$
and $\forall x \in [x_0;x_1).\ f(x) \geq 0$
and $\forall x \geq x_1.\ f(x) = g(x) + \sum_{i=1}^{k} as_i \cdot f(ts_i(x))$
and $\forall x \geq x_1.\ g(x) \geq 0$

Fig. 1 The locale *akra_bazzi_function* that formally captures the conditions imposed upon the recursively-defined function f

development in the *Archive of Formal Proofs* [9] or, where applicable, Leighton's proof [14]. In any case, we recommend that readers familiarise themselves with Leighton's proof before attempting to understand ours.

7.1 Formal Setting

First of all, we will explain how the conditions mentioned in Sect. 4 are stated formally in Isabelle/HOL: We use a *locale* [4] called *akra_bazzi_function*. A locale is a named context that contains fixed variables, assumptions, and definitions. Such a locale can be *instantiated* by providing values for the fixed variables and proving that its assumptions hold for these values. Instantiating a locale gives the user access to all the facts that were proven in this locale, specialised to the specific values that it was instantiated with. Figure 1 shows the definition of the locale *akra_bazzi_function*, modulo some insignificant notational adjustments.

The only difference to the conditions stated in Sect. 4 is that the recursive calls are of the shape $f(ts_i(x))$ instead of $f\ (b_i \cdot x + h_i\ x)$. The reason for this is that the latter would require expressing a call like $f(\lfloor \frac{1}{2}x \rfloor)$ in the rather awkward form $f(\frac{1}{2}x + (\lfloor \frac{1}{2}x \rfloor - \frac{1}{2}x))$, whereas the former is more direct.

The conditions on the recursive calls are replaced by the condition that all the ts_i be Akra–Bazzi terms, where

definition $\text{akra\_bazzi\_term}(x_0, x_1, b, t) =$

$$(\exists \varepsilon\ h.\ \varepsilon > 0\ \wedge\ h \in O(\lambda x.\ x / \ln^{1+\varepsilon} x)\ \wedge$$
$$(\forall x \geq x_1.\ t(x) \geq x_0\ \wedge\ t(x) < x\ \wedge\ b \cdot x + h(x) = t(x)))\ .$$

One can then easily prove introduction rules to discharge this condition for specific forms of recursive calls, e. g.

lemma akra_bazzi_term_ceiling:
assumes $b > 0$ **and** $b < 1$ **and** $x_0 \leq b \cdot x_1$ **and** $(1 - b) \cdot x_1 \geq 1$
shows $\text{akra\_bazzi\_term}(x_0, x_1, b, \lambda x.\ \lceil b \cdot x \rceil)$

Provided that such rules exist for every recursive call occurring in the recursive equation of f, this condition and most of the other locale assumptions contain only the constants *as*, *bs*, k, x_0, and x_1 and can therefore be solved by simple evaluation for a concrete function f with concrete values for *as*, *bs*, etc. The remaining conditions are:

1. $\forall x \in [x_0; x_1).\ f(x) \geq 0$
2. $\forall x \geq x_1.\ g(x) \geq 0$
3. $\forall x \geq x_1.\ f(x) = g(x) + \sum_{i=1}^{k} as_i \cdot f\ (ts_i(x))$

These conditions must be shown by the user, but they are typically direct consequences from the definitions of f and g.

7.2 Asymptotic Estimates

We now move on to the actual proofs. First of all, we need to prove a number of asymptotic inequalities, i.e. inequalities that hold whenever x is large enough. Leighton mentions the first four of these on page 5, but does not provide any proof (he mentions that they can be proven using 'standard Taylor series expansions and asymptotic analysis'). Apart from these, we found that we need four more inequalities that ensure that conditions like $b_i x + h_i(x) < x$ and $1 - \ln(b_i x + h_i(x))^{-\varepsilon/2} > 0$ will hold for all relevant inputs x. Without these conditions, several terms that occur in the proof of the Akra–Bazzi theorem would not even be well-defined.

Lemma 1 (Asymptotic inequalities) *For any $H, \varepsilon \in \mathbb{R}_{>0}$, $b \in (0; 1)$, and $p \in \mathbb{R}$, there exists some $x_0 \in \mathbb{R}$ such that the following inequalities hold for all $x \geq x_0$:*

$$\left(1 \pm \frac{H}{b \ln^{1+\varepsilon} x)}\right)^p \left[1 + \left(\ln^{-\varepsilon/2}\left(bx + \frac{Hx}{\ln^{1+\varepsilon} x}\right)\right)\right] \geq 1 + \ln^{-\varepsilon/2} x \tag{1}$$

$$\left(1 \pm \frac{H}{b \ln^{1+\varepsilon} x}\right)^p \left[1 - \left(\ln^{-\varepsilon/2}\left(bx + \frac{Hx}{\ln^{1+\varepsilon} x}\right)\right)\right] \leq 1 - \ln^{-\varepsilon/2} x \tag{2}$$

$$\frac{1}{2}\left(1 + \ln^{-\varepsilon/2} x\right) \leq 1 \tag{3}$$

$$2\left(1 - \ln^{-\varepsilon/2} x\right) \geq 1 \tag{4}$$

$$\left[\ln\left(bx - \frac{Hx}{\ln^{1+\varepsilon} x}\right)\right]^{-\varepsilon/2} < 1 \tag{5}$$

$$\frac{H}{\ln^{1+\varepsilon} x} < \frac{b}{2} \tag{6}$$

$$\frac{H}{\ln^{1+\varepsilon} x} < \frac{1-b}{2} \tag{7}$$

$$x\left(1 - b - \frac{H}{\ln^{1+\varepsilon} x}\right) > 1 \tag{8}$$

(The $\pm$ means that the inequality must hold both for a $+$ and for a $-$ in its place.)

Proof All but the first two of these inequalities are trivial and can be proven by comparing the limits of the left-hand side and the right-hand side; the first two, however, require non-trivial asymptotic analysis using Taylor series expansions. Since these two inequalities are a crucial ingredient in the proof of this generalised Akra–Bazzi theorem, we will briefly sketch the proof of (1). (The proof of (2) is mostly analogous.)

The key ingredient in proving the inequality is the Taylor series expansion

$$(1 \pm t(x))^y = 1 \pm yt(x) + O\left(t(x)^2\right) = 1 + O(t(x)) \qquad \text{if } \lim_{x\to\infty} t(x) = 0$$

In the following, we will indicate such an expansion with the symbol $\overset{\text{Taylor}}{=}$ and a curly bracket that denotes which term is taken to be $t(x)$ in the expansion.

First of all, we estimate the first factor on the left-hand side with[4]

$$\left(1 \pm \overbrace{\frac{H}{b\ln^{1+\varepsilon} x}}^{t(x)}\right)^p \overset{\text{Taylor}}{=} 1 + O\left(\ln^{-1-\varepsilon} x\right) = 1 + o\left(\ln^{-1-\varepsilon/2} x\right)$$

Moreover, we have in the second factor:

$$\begin{aligned}
&\ln^{-\varepsilon/2}\left(bx + \frac{Hx}{\ln^{1+\varepsilon} x}\right)\\
&= \left[\ln\left(bx\left(1 + \frac{H}{b\ln^{1+\varepsilon} x}\right)\right)\right]^{-\varepsilon/2}\\
&= \left[\ln bx + \ln\left(1 + \frac{H}{b\ln^{1+\varepsilon} x}\right)\right]^{-\varepsilon/2}\\
&= \left(\ln^{-\varepsilon/2} bx\right)\left[1 + \underbrace{\frac{1}{\ln bx}\ln\left(1 + \frac{H}{b\ln^{1+\varepsilon} x}\right)}_{t(x)}\right]^{-\varepsilon/2} \overset{\text{Taylor}}{=}\\
&= \left(\ln^{-\varepsilon/2} bx\right)\left[1 + O\left(\frac{1}{\ln bx}\ln\left(1 + \frac{H}{b\ln^{1+\varepsilon} x}\right)\right)\right]\\
&= \left(\ln^{-\varepsilon/2} bx\right) + \left(\ln^{-1-\varepsilon/2} bx\right) O\left(\ln\left(1 + \frac{H}{b\ln^{1+\varepsilon} x}\right)\right)\\
&= \left(\ln^{-\varepsilon/2} bx\right) + \left(\ln^{-1-\varepsilon/2} bx\right) o(1)\\
&= \left(\ln^{-\varepsilon/2} bx\right) + o\left(\ln^{-1-\varepsilon/2} x\right)
\end{aligned}$$

Combining these two asymptotic estimates, we have:

$$\begin{aligned}
&\left(1 \pm \frac{H}{b\ln^{1+\varepsilon} x}\right)^p \left[1 + \left(\ln^{-\varepsilon/2}\left(bx + \frac{Hx}{\ln^{1+\varepsilon} x}\right)\right)\right]\\
&= \left[1 + o\left(\ln^{-1-\varepsilon/2} x\right)\right]\left[1 + \left(\ln^{-\varepsilon/2} bx\right) + o\left(\ln^{-1-\varepsilon/2} x\right)\right]\\
&= 1 + (\ln bx)^{-\varepsilon/2} + o\left(\ln^{-1-\varepsilon/2} x\right)\\
&= 1 + (\ln b + \ln x)^{-\varepsilon/2} + o\left(\ln^{-1-\varepsilon/2} x\right)\\
&= 1 + \left(\ln^{-\varepsilon/2} x\right)\left(1 + \underbrace{\frac{\ln b}{\ln x}}_{t(x)}\right)^{-\varepsilon/2} + o\left(\ln^{-1-\varepsilon/2} x\right) \overset{\text{Taylor}}{=}\\
&= 1 + \left(\ln^{-\varepsilon/2} x\right)\left(1 - \frac{\varepsilon\ln b}{2\ln x} + O\left(\ln^{-2} x\right)\right) + o\left(\ln^{-1-\varepsilon/2} x\right)\\
&= 1 + \ln^{-\varepsilon/2} x + \left[\frac{-\varepsilon\ln b}{2}\ln^{-1-\varepsilon/2} x + o\left(\ln^{-1-\varepsilon/2} x\right)\right]
\end{aligned}$$

[4] The notation here becomes a bit informal. Terms like $f(x) + O(g(x))$ stand for the set $\{f(x) + h(x) \mid h(x) \in O(g(x))\}$ and all equality symbols are then essentially set inclusions, i.e. $f(x) + O(\ldots) = g(x) + O(\ldots)$ means that the left-hand side is a subset of the right-hand side.

Since $b \in (0; 1)$, we have $\ln b < 0$. Therefore, the term in brackets will be positive for sufficiently large x and we have:

$$\geq 1 + \ln^{-\varepsilon/2} x$$

□

7.3 The Continuous Akra–Bazzi Theorem

For the next part, we closely follow Leighton's proof (pp. 6–8). Here we prove the Akra–Bazzi theorem for *continuous* recurrences, i. e. a function $f : \mathbb{R} \to \mathbb{R}$ that fulfils all the conditions stated before, but not just on $\mathbb{N}$, but on $\mathbb{R}$. We therefore look at the following setting, which is essentially the real-valued analogue to the setting described in Sect. 4:

Consider $f : \mathbb{R} \to \mathbb{R}$ with the following properties:

$$f(x) \geq 0 \qquad \text{for all } x \in [x_0; x_1]$$

$$f(x) = g(x) + \sum_{i=1}^{k} a_i \cdot f(b_i \cdot x + h_i(x)) \qquad \text{for all } x > x_1$$

for a natural number $k \in \mathbb{N} \setminus \{0\}$, a function $g : \mathbb{R} \to \mathbb{R}$, natural numbers $x_0, x_1 \in \mathbb{R}$, real coefficients $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$, functions $h_i : \mathbb{N} \to \mathbb{R}$, and $p \in \mathbb{R}$ such that:

- $g(x) \geq 0$ for all $x \geq x_0$
- $a_i \geq 0$ for all $i \in [1; k]$ and $a_i > 0$ for at least one $i \in [1; k]$
- $b_i \in (0; 1)$ for all $i \in [1; k]$
- for every $i \in [1; k]$, there exists an $\varepsilon_i > 0$ such that $h_i \in O(x/\ln^{1+\varepsilon_i} x)$
- $\sum_{i=1}^{k} a_i \cdot b_i^p = 1$
- $g(u)u^{-p-1}$ is integrable on $[x_0; x]$ for any $x \geq x_0$

We can assume w.l.o.g. that all the h_i fulfil $h_i \in O(x/\ln^{1+\varepsilon} x)$ for the same ε by choosing the minimum of all the ε_i values. It is then clear that there exists a constant H such that, for sufficiently large x, we have $|h_i(x)| \leq Hx \ln^{-1-\varepsilon} x$ for all $i \in [1; k]$.

We also assume that x_0 and x_1 are chosen large enough such that the following inequalities hold:

- $1 \leq x_0 \leq \frac{1}{2} b_i x_1$ for all $i \in [1; k]$
- $|h_i(x)| \leq Hx \ln x^{-1-\varepsilon}$ for all $i \in [1; k]$ and all $x \geq x_1$
- the inequalities (1) to (8) from Lemma 1 for any $b \in \{b_1 \dots b_k\}$ and all $x \geq x_0$
- there exists some real number C such that $Cx \leq b_i x - Hx \ln^{-1-\varepsilon} x$ for any $i \in [1; k]$ and all $x \geq x_1$

7.3.1 The Lower Bound

We will now show how to obtain an asymptotic lower bound on $f(x)$. For this, we further have to assume the existence of positive and finite bounds

$$F := \inf_{x \in [x_0; x_1]} f(x) \qquad \text{and} \qquad G := \sup_{x \in [x_0; x_1]} g(x)$$

and the growth condition $\forall x \geq x_1. \forall u \in [Cx; x]. c_2 g(x) \geq g(u)$ for some $c_2 > 0$. We can then show the following two lemmas:

Lemma 2 *There exists a* $c_4 > 0$ *such that, for any* $i \in [1; k]$ *and all* $x \geq x_1$*:*

$$x^p \int_{b_i x + h_i(x)}^{x} \frac{g(u)}{u^{p+1}}\, du \leq c_4 g(x)$$

Proof Technical and uninteresting; refer to Leighton's Lemma 1 or our formal proof development for details.

Lemma 3 *There exists a* $c_5 > 0$ *with* $c_5 \leq \frac{1}{2c_4}$ *such that, for any* $x \in [x_0; x_1]$*:*

$$2c_5 x^p \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, du\right) \leq f(x)$$

Proof We have $x^p \leq \max(x_0^p, x_1^p)$ and $g(u)u^{-p-1} \leq G \cdot \max(x_0^{-p-1}, x_1^{-p-1})$; it is easy to use this to derive some bound c such that

$$\frac{2}{F} x^p \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \leq c$$

and therefore

$$\frac{2}{c} x^p \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \leq F \leq f(x).$$

Setting $c_5 := \min(c^{-1}, (2c_4)^{-1})$ yields the desired bound. □

We can then show the following lower bound for f

Lemma 4

$$c_5 x^p \left(1 + \ln^{-\varepsilon/2} x\right) \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, du\right) \leq f(x) \quad \textit{for all } x \geq x_0$$

Proof The proof is by induction over x with the base case $x \in [x_0; x_1]$ and the inductive step $x > x_1$ while assuming that the induction hypothesis holds for all $b_i x + h_i(x)$ for any $i \in [1; k]$. This induction scheme is well-founded, since the bounds on the h_i and inequality (8) imply $\lceil b_i x + h_i(x) \rceil < \lceil x \rceil$ for any $x \geq x_1$, so the measure $\mu : \mathbb{R} \to \mathbb{N}, x \mapsto \lceil x \rceil$ decreases.

Base case We have $x \in [x_0; x_1]$ and therefore:

$$c_5 x^p \underbrace{\left(1 + \ln^{-\varepsilon/2} x\right)}_{\substack{\leq 2 \\ \text{by Lemma 1.(3)}}} \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \leq 2c_5 x^p \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \overset{\text{Lemma 3}}{\leq} f(x)$$

Induction step We have $x > x_1$ and we assume the following induction hypothesis for any $i \in [1; k]$:

$$c_5 (b_i x + h_i(x))^p \left(1 + \ln^{-\varepsilon/2}(b_i x + h_i(x))\right) \left(1 + \int_{x_0}^{b_i x + h_i(x)} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \qquad \text{(IH)}$$
$$\leq f(b_i x + h_i(x))$$

We can then show:

$$c_5 x^p \left(1 + \ln^{-\varepsilon/2} x\right) \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right) \overset{\text{Lemma 3}}{\leq}$$

$$
\begin{aligned}
&\le c_5 x^p \left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)+g(x)-2c_5c_4 g(x) \overset{\text{Lemma 1.(3)}}{\le}\\
&\le c_5 x^p \left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)+g(x)-c_5c_4\left(1+\ln^{-\varepsilon/2} x\right)g(x)\\
&= g(x)+c_5x^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u-\frac{c_4}{x^p}g(x)\right)\overset{\text{Lemma 2}}{\le}\\
&= g(x)+c_5x^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u-\int_{b_ix+h_i(x)}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\\
&= g(x)+c_5x^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{b_ix+h_i(x)}\frac{g(u)}{u^{p+1}}\mathrm{d}u\right)\\
&= g(x)+\left(\sum_{i=1}^{k}a_ib_i^p\right)c_5x^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{b_ix+h_i(x)}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\\
&= g(x)+\sum_{i=1}^{k}a_ic_5b_i^px^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{b_ix+h_i(x)}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)
\end{aligned}
$$

Let us now focus on the $b_i^p x^p \left(1+\ln^{-\varepsilon/2} x\right)$ term. Let $s{:=}-1$ if $p \ge 0$ and $s{:=}1$ otherwise. By applying first Lemma 1.(1) and then $|h_i(x)| \le Hx\ln x^{-1-\varepsilon}$, we have:

$$
\begin{aligned}
b_i^px^p\left(1+\ln^{-\varepsilon/2} x\right) &\overset{(1)}{\le} b_i^px^p\left(1+\frac{sH}{b_i\ln^{1+\varepsilon} x}\right)^p\left(1+\ln^{-\varepsilon/2}\left(b_ix+\frac{Hx}{\ln^{1+\varepsilon} x}\right)\right)\\
&= \left(b_ix+\frac{sHx}{\ln^{1+\varepsilon} x}\right)^p\left(1+\ln^{-\varepsilon/2}\left(b_ix+\frac{Hx}{\ln^{1+\varepsilon} x}\right)\right)\\
&\le (b_ix+h_i(x))^p\left(1+\ln^{-\varepsilon/2}(b_ix+h(x))\right)
\end{aligned}
$$

Plugging this result into the inequality chain we interrupted before, we have:[5]

$$
\begin{aligned}
&g(x)+\sum_{i=1}^{k}a_ic_5b_i^px^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{b_ix+h_i(x)}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\\
&\quad\le g(x)+\sum_{i=1}^{k}a_ic_5\left(b_ix+h_i(x)\right)^p\left[1+\ln^{-\varepsilon/2}(b_ix+h(x))\right]\left(1+\int_{x_0}^{b_ix+h_i(x)}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\overset{(IH)}{\le}\\
&\quad\le g(x)+\sum_{i=1}^{k}a_i f(b_ix+h_i(x))=f(x)
\end{aligned}
$$

This concludes the induction. We have thus shown that for all $x \ge x_0$:

$$
c_5x^p\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\le c_5x^p\left(1+\ln^{-\varepsilon/2} x\right)\left(1+\int_{x_0}^{x}\frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\le f(x)
$$

□

[5] Note the implicit case distinction for $p \ge 0$ and $p < 0$: for $p \ge 0$, we need to show $b_ix - Hx\ln^{-1-\varepsilon} x \le b_ix + h_i(x)$, whereas for $p < 0$ we need to show $b_ix + Hx\ln^{-1-\varepsilon} x \ge b_ix + h_i(x)$ since the negative exponent flips the inequality. This case distinction is not mentioned by Leighton.

7.3.2 The Upper Bound

The proof of the upper bound is analogous with the following exceptions:

- The assumptions we need are $\sup_{x\in[x_0;x_1]} f(x) < \infty$ and $\forall x \geq x_1.\ \forall u \in [Cx; x].\ c_1 g(x) \leq g(u)$ for some $c_1 > 0$.
- The inequality we need to show by induction is

$$f(x) \leq c_6 x^p \left(1 - \ln^{-\varepsilon/2} x\right) \left(1 + \int_{x_0}^{x} \frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)$$

7.4 Lifting to the Discrete Case

The remaining work in the proof of the Akra–Bazzi theorem is now to lift this result for continuous recurrences to discrete recurrences. We will again illustrate this for the lower bound. We consider the setting given in Theorem 1.

First of all, we find values $G > 0$ and $\hat{x}_0, \hat{x}_1$ such that:

- $x_1 \leq \hat{x}_0 \leq \hat{x}_1$
- $f(x) > 0$ for all $x \geq \hat{x}_0$
- $g(x) \geq G\bar{g}(x)$ for all $x \geq \hat{x}_0$ (possible because $g \in \Omega(\bar{g})$)
- $\bar{g}(x) \geq 0$ for all $x \geq \hat{x}_0$
- $b_i x + h_i(x) \geq \hat{x}_0$ for all $x \geq \hat{x}_1$ and $i \in [1; k]$

Since g has the asymptotic lower bound $\bar{g}$ and we want to find an asymptotic lower bound on f, it is natural that the function obtained by copying the definition of f, but with the costs $\bar{g}$ instead of g, should be an asymptotic lower bound on f. We therefore define the function $\hat{f} : \mathbb{N} \to \mathbb{R}$ as follows:

$$\hat{f}(x) = \begin{cases} \max(0, f(x)/G) & \text{if } x < \hat{x}_1 \\ \bar{g}(x) + \sum_{i=1}^{k} a_i \hat{f}(b_i x + h_i(x)) & \text{otherwise} \end{cases}$$

It is clear that $\hat{f}$ is non-negative everywhere, positive for large enough x, and $G\hat{f}(x) \leq f(x)$ for all $x \geq x_0$. Therefore, any asymptotic lower bound on $\hat{f}$ will also be a lower bound on f.

Next, we find some $\bar{x}_0 > \hat{x}_1$ such that

- the asymptotic inequalities (1) to (8) hold for all $x \geq x_0$, $c = H$, $b = b_i$ for $i \in [1; k]$
- $|h_i(x)| \leq Hx \ln^{-1-\varepsilon} x$ for all $x \geq \bar{x}_0$ and $i \in [1; k]$
- $Cx \leq b_i x - Hx \ln^{-1-\varepsilon} x$ for all $x \geq \bar{x}_0$ and $i \in [1; k]$
- $\bar{g}$ is bounded from above on all intervals that lie above $\bar{x}_0$
- $\bar{g}(u) \leq c_2 \bar{g}(x)$ for all $x \geq \bar{x}_0$ and $u \in [Cx; x]$
- $\hat{f}(\lfloor x \rfloor) > 0$ and $\bar{g}(x) \geq 0$ for all $x \geq \bar{x}_0$
- $\bar{g}(u)u^{-p-1}$ is integrable on all intervals that lie above $\bar{x}_0$

and we find a $\bar{x}_1$ such that $\bar{x}_1 \geq \frac{2}{b_i}\bar{x}_0$ for all $i \in [1; k]$.

We can then extend $\hat{f}$ to a function $\bar{f} : \mathbb{R} \to \mathbb{R}$ using the following definition:

$$\bar{f}(x) = \begin{cases} \hat{f}(\lfloor x \rfloor) & \text{if } x \leq \bar{x}_1 \\ g(x) + \sum_{i=1}^{k} a_i \bar{f}(b_i x + h_i(x)) & \text{otherwise} \end{cases}$$

Clearly, $\bar{f}(x) = \hat{f}(x)$ for all natural numbers x.

We apply the continuous lower bound theorem derived in the previous section to $\bar{f}(x)$ with the 'base cases' between $\bar{x}_0$ and $\bar{x}_1$. This gives us the lower bound

$$c_5 x^p \left(1 + \int_{\bar{x}_0}^{x} \frac{\bar{g}(u)}{u^{p+1}}\,\mathrm{d}u\right) \le \bar{f}(n)$$

For all natural numbers $n \ge \bar{x}_0$, we then have:

$$G c_5 n^p \left(1 + \int_{\bar{x}_0}^{n} \frac{\bar{g}(u)}{u^{p+1}}\,\mathrm{d}u\right) \le G\bar{f}(n) = G\hat{f}(n) \le f(n)$$

and therefore

$$f \in \Omega\left(n^p \left(1 + \int_{\bar{x}_0}^{n} \frac{\bar{g}(u)}{u^{p+1}}\,\mathrm{d}u\right)\right).$$

Since we could have chosen $\bar{x}_0$ larger as well if we had wanted to, we have essentially shown that

$$f \in \Omega\left(n^p \left(1 + \int_{t}^{n} \frac{\bar{g}(u)}{u^{p+1}}\,\mathrm{d}u\right)\right)$$

holds for any sufficiently large t.

The formal proof of all of this consists mostly of finding large enough values for $\bar{x}_0$ and $\bar{x}_1$ and proving that all required properties hold. For instance, the condition that $\inf_{x\in[\bar{x}_0;\bar{x}_1]} \bar{f}(x) > 0$ is fulfilled because, by definition, $\bar{f}(x) = \hat{f}(\lfloor x \rfloor)$ only takes a finite number of values on the interval $[\bar{x}_0; \bar{x}_1]$, all of which are positive.

This entire process of linking the assumptions of the discrete settings to the assumptions of the continuous setting is very technical and intricate—it takes up almost a quarter of the entire proof of the Akra–Bazzi theorem—but mathematically uninteresting, which is why we left out a lot of detail in this paper proof.

7.5 The Master Theorem

From the proof of the Akra–Bazzi theorem in the locale context *akra_bazzi_function*, we can now show the Master Theorem, also inside this context. The proofs are straightforward applications of the Akra–Bazzi theorem. The user can then interpret the *akra_bazzi_function* locale for her function f and use the case of the Master Theorem appropriate for her function.

In the Isabelle formalisation, the Master Theorem is split into five cases (cf. Table 2), with the first case having a weak form (O) and a strong form (Θ).

Table 2 The five cases of the Master Theorem as formalised in Isabelle/HOL

Case name	Assumptions			Conclusion
Case 1 (O)	$g \in O(x^q)$	$q < p$		$f \in O(x^p)$
Case 1	$g \in O(x^q)$	$q < p$	$f(x) > 0$ (for suff. large x)	$f \in \Theta(x^p)$
Case 2.1	$g \in \Theta(x^p \ln^q x)$	$q < -1$		$f \in \Theta(x^p)$
Case 2.2	$g \in \Theta(x^p / \ln x)$			$f \in \Theta(x^p \ln\ln x)$
Case 2.3	$g \in \Theta(x^p \ln^{q-1} x)$	$q > 0$		$f \in \Theta(x^p \ln^q x)$
Case 3	$g \in \Theta(x^q)$	$q > p$		$f \in \Theta(x^q)$

8 Automation

The formalisation also contains three proof methods that add a certain degree of automation to the usage of the Master Theorem. We will describe them in the following sections.

8.1 Akra–Bazzi Terms

As mentioned previously in Sect. 7.1, the condition that recursive calls must be Akra–Bazzi terms can be discharged by introduction rules that reduce the condition to simple statements on constants. We provide a theorem collection called `akra_bazzi_term_intro` to which the user can add custom introduction rules for Akra–Bazzi terms. The Akra–Bazzi proof methods then automatically use these rules to discharge conditions of this kind.

8.2 Akra_Bazzi_Termination

It is possible to define recursive functions with complex recursion schemes in Isabelle/HOL [13]. For every function definition, the user must show that the function is indeed *well-defined*: the definition must be complete, different equations must not overlap with one another, and the function must terminate, i. e. there must not be any infinite chains of recursive calls. The first two conditions can virtually always be proven automatically; the last condition, termination, is usually the most difficult to prove. Isabelle/HOL can prove it automatically in many cases, but Akra–Bazzi style recursion schemes are usually not among them: for example, attempting to define the cost function for Merge Sort recursively by its recurrence relation $f(n) = f(\lfloor \frac{n}{2} \rfloor) + f(\lceil \frac{n}{2} \rceil) + n$ for $n \geq 2$ will fail since Isabelle's termination prover is unable to prove that n becomes smaller in every recursion step.

To aid the user in proving termination for such recursion schemes, we developed the proof method `akra_bazzi_termination`, which uses the `akra_bazzi_term_intro` rules mentioned before to reduce the proof obligation of termination to simpler proof obligations that contain only constants, which can then be solved automatically using Isabelle's simplifier. A typical function definition of an Akra–Bazzi function then looks like this:

function merge_sort_cost::nat $\Rightarrow$ real **where**

merge_sort_cost(0) $= 0$

| merge_sort_cost(1) $= 1$

| $n \geq 2 \Longrightarrow$ merge_sort_cost(n) $=$

merge_sort_cost($\lfloor n/2 \rfloor$) $+$ merge_sort_cost($\lceil n/2 \rceil$) $+ n$

The termination of this function can be proven using the `akra_bazzi_termination` method. In this case, it produces ten proof obligations of the form $0 < \frac{1}{2}, \frac{1}{2} < 1, 0 \leq \frac{1}{2} \cdot 2$, etc. These can be solved automatically with the simplifier.

It should be noted that `akra_bazzi_termination` works in more complicated situations as well, e. g. when the function has several arguments (curried or tupled), which may even change during the recursive call.

In order to achieve this, `akra_bazzi_termination` performs some analysis of the function's type to find a parameter that is a natural number and for which the recursive calls are Akra–Bazzi terms. It then starts a termination proof using this parameter as a termination measure and applies the corresponding introduction rules for Akra–Bazzi terms, which leaves only the simple proof obligations we saw earlier.

Note also that `akra_bazzi_termination` is completely independent from the Akra–Bazzi theorem itself; its only connection to Akra–Bazzi is the fact that it helps automate termination proofs of Akra–Bazzi-style recurrences.

8.3 Master_Theorem

The main proof method is `master_theorem`, which can be invoked on goals of the form $f \in O(\_)$ or $f \in \Theta(\_)$. It takes as an argument the applicable case of the Master Theorem (e. g. 1 or 2.2) and applies it to the goal. The recursive equation of f and the values x_0 and x_1 are optional parameters which the method attempts to guess if absent. The appropriate values for k, as, bs, ts, and g are always inferred automatically from the recursive equation, provided the required `akra_bazzi_term_intro` rules exist.
To provide some more detail as to where all these values come from:

- The function f can obviously be determined from the goal itself.
- If not provided explicitly, the method will try to obtain the recursive equation for f from its definition.
- p, the characteristic number of the recurrence, never needs to be specified explicitly. If it appears in the goal, the proof method infers it from the goal; if it does not appear in the goal, its value is not required explicitly – if there is e. g. a proof obligation that some number q is larger than p, the user is presented with the equivalent proof obligation that $\sum_{i=1}^{k} a_i \cdot b_i^q < 1$.
- x_0 is set to 0 if not given explicitly, which is always a correct choice except in the unusual case that f is negative for some inputs.
- x_1 is determined from the precondition of the recursive equation (e. g. 2 for the precondition $n > 1$).
- k, as and ts are determined by partitioning the right-hand side of the recursive equation into summands and bringing each summand into the form $a_i \cdot f(ts_i\ n)$ if possible.
- g is the sum of all summands that cannot be brought into this form.
- for each $t \in ts$, the corresponding $b \in bs$ is determined by finding a rule from the theorem collection `akra_bazzi_term_intro` that matches t and extracting b from the unifier.

We will now illustrate the practical application of the proof method with two examples.

Example: Merge Sort As an example, we consider the `merge_sort_cost` function from Section 8.2. Case 2.3 of the Master Theorem tells us that the growth of this function is $\Theta(n \ln n)$. In Isabelle, we write:

lemma merge_sort_cost $\in \Theta(\lambda n.\ n \cdot \ln\ n)$
apply (master_theorem 2.3)

This leaves us with the following proof obligations:

1. $\forall n.\ 0 \le n \implies n < 2 \implies 0 \le$ merge_sort_cost n
2. $\forall n.\ 2 \le n \implies$ merge_sort_cost$(n) = n +$ merge_sort_cost$(\lfloor n/2 \rfloor) +$ merge_sort_cost$(\lceil n/2 \rceil)$
3. $\forall n.\ 2 \le n \implies 0 \le n$
4. $(\lambda n.\ n) \in \boldsymbol{\Theta}(\lambda n.\ n)$

The first goal, non-negativity of merge_sort_cost, can be proven easily by case distinction using the function definition. All the remaining goals can be discharged automatically by the simplifier.

Example: Boncelet coding Another interesting example is given by Drmota and Szpankowski [8]: the average phrase length $d(n)$ in Boncelet coding [6]. This is not related to Divide and Conquer algorithms at all, but d does fulfil the Akra–Bazzi-type recurrence

$$d(n) = 1 + p \cdot d(\lfloor p \cdot n + \delta \rfloor) + q \cdot d(\lfloor q \cdot n - \delta \rfloor)$$

where $p \in (0; 1)$ is a probability, $q = 1 - p$, and $\delta > 0$, $\delta < 1$, $2p + \delta < 2$. Since our Master Theorem applies to this recurrence, we can use the `master_theorem` tactic:

lemma boncelet_phrase_length:

fixes	$p\ \delta$::real
fixes	d::nat $\Rightarrow$ real
defines	$q \equiv 1 - p$
assumes	$p \in (0; 1)$ **and** $\delta \in (0; 1)$ **and** $2 \cdot p + \delta < 2$ **and** $\forall n.\ d(n) \geq 0$
assumes	$\forall n \geq 2.\ d(n) = 1 + p \cdot d(\lfloor p \cdot n + \delta \rfloor) + q \cdot d(\lfloor q \cdot n + \delta \rfloor)$
shows	$d \in \Theta(\lambda x.\ \ln x)$

using assms **by** (master_theorem recursion: d_rec, simp_all)

This example is particularly interesting because here we do not have a concrete function that we defined ourselves, but prove a theorem for an entire class of functions satisfying a certain recurrence. Since d is not a function-package function, we need to give `master_theorem` the recurrence rule to use, but then, the lemma can be proven automatically again. Note also that we did not specify which case to use here; Isabelle implicitly backtracks over all possible cases and succeeds in case 2.3. It should, however, be mentioned that the Θ-estimate is not very useful in this case: as Drmota and Szpankowski elaborate, an analysis of the redundancy of the Boncelet scheme requires more precise asymptotics, like the ones they provide.

Further examples of complexity proofs for recurrences derived from the computational costs of Divide and Conquer algorithms such as Karatsuba's multiplication algorithm for natural numbers, Strassen's algorithm for matrix multiplication, or the deterministic Median-of-Medians selection algorithm can be found in the examples file of our entry in the *Archive of Formal Proofs* [9].All examples can be proven mostly automatically with the `master_theorem` method and Isabelle's simplifier. (in no small part thanks to the decision procedure we presented in Sect. 3.2.2) There are only three kinds of subgoals that are sometimes left behind:

– Positivity/non-negativity for sufficiently large inputs; this can typically be shown by straightforward induction and simplification.
– Showing that p satisfies $\sum_{i=1}^{k} as_i \cdot bs_i^p = 1$; proving this is usually automatic, but can be non-trivial in some cases. One such example is that from our introduction: $f(n) = f(\lfloor n/4 \rfloor) + f(\lfloor n/2 \rfloor) + 1$ with $p = \log_2 \varphi$, where φ is the golden ratio.
– Inequalities like $q < p$ for complicated constants p, q (e. g. involving logarithms and square roots); these can usually be proven automatically by approximation [12].

8.4 Akra_Bazzi_Approximation

In some of the cases of the Master Theorem, the goal contains the parameter p. As shown before, this p always exists and is unique. It can, however, not always be expressed in a closed form. One example for such a situation is the function $f(x) = f(\lfloor x/3 \rfloor) + f(\lfloor 3x/4 \rfloor)$, for which p seems to have no closed form, but can be approximated to 1.152.

Our Master Theorem in Isabelle/HOL yields

$$\textbf{lemma}\ \ f \in \Theta\left(\lambda n.\ n^{\text{akra\_bazzi\_exponent}([1,1],[1/3,3/4])}\right)$$

where *akra_bazzi_exponent* is a function that, given lists *as* and *bs* that fulfil the usual conditions, returns the unique p such that $\sum_{i=1}^{k} as_i \cdot bs_i^p = 1$.

Of course, one would now like to obtain and verify at least an approximate value for this exponent. An approximate value for p can be found e.g. using an external computer algebra system or by applying Newton's method as described in Sect. 8.4. Once such an approximation has been found, it can then be verified with our proof method `akra_bazzi_approximate`:

$$\textbf{lemma}\ \text{akra\_bazzi\_exponent}([1,1],[1/3,\ 3/4]) \in [1.1519623;\ 1.1519624]$$
$$\textbf{by}\ (\text{akra\_bazzi\_approximate } 29)$$

This proof method internally uses the `approximation` tactic [12], which uses Taylor series expansions, interval arithmetic, and reflection (i.e. Isabelle's code generator [11]) to certify bounds on the values of certain transcendental functions. The number 29 in the invocation here indicates the precision (measured in binary digits) of the computation. If it is too low, the proof attempt might fail even though the statement is true; if it is too high, the proof will take more time. The user of the proof method should therefore find a precision value that is as low as possible while still being high enough for the proof to succeed.

Like `akra_bazzi_termination`, the method `akra_bazzi_approximation` is completely orthogonal to the Akra–Bazzi theorem; neither depends on the other. In particular, no part of the proof of the Akra–Bazzi theorem relies on numerical approximation.

Approximating p As a side note: p can be approximated quite easily. Recall that p was defined as the unique real number such that $t(p) = 1$ where

$$t : \mathbb{R} \to \mathbb{R}_{>0},\ x \mapsto \sum_{i=1}^{k} a_i \cdot b_i^x.$$

Note that $t(x) - 1$ is convex and $t'(x)$ has no zeros. This means that Newton's method can be used to approximate p very efficiently: Due to the convexity of $t(x) - 1$, a Newton step at any lower bound (i.e. intersecting the tangent of t at the lower bound with the x axis) will yield a better lower bound, and intersecting the secant of a lower bound and an upper bound on p with the x axis will yield a better upper bound. This can be used to obtain good approximation intervals for p very quickly.

As initial lower and upper bounds, one can e.g. use the estimates

$$-\frac{\ln a_k}{\ln b_k} \le p \le -\frac{\ln\left(n \cdot \max_{i\in[1;n]} a_i\right)}{\ln\left(\max_{i\in[1;n]} b_i\right)} \qquad \text{where } k \in [1;n] \text{ arbitrary.}$$

We have not verified this approximation algorithm in Isabelle, since the approximation can easily be done externally in an unverified way and then certified using e.g. the `akra_bazzi_approximation` method.

9 Comparison with Similar Theorems

9.1 Leighton's Akra–Bazzi Theorem

Our version of the Akra–Bazzi theorem differs from Leighton's [14] in a few minor respects:

- We require only one a_i to be positive and the rest to be non-negative. In Leighton's version, all of them must be positive.
- Every property that we require to hold between x_0 and x_1, Leighton requires to hold between 1 and x_0. (which corresponds to our x_1)
- We have two separate functions g and $\bar{g}$, where the latter is an integrable asymptotic bound for the former. Leighton essentially requires g and $\bar{g}$ to be the same.
- We have not only a Θ version of the Akra–Bazzi theorem, but also O and Ω versions, which may be useful when the behaviour of g is not fully known.

9.2 Master Theorem

Due to its being derived from the Akra–Bazzi theorem, our version of the Master Theorem is much more general than the versions of the Master Theorem that are typically presented in the literature (e. g. *Introduction to Algorithms* [7]). Our version of the Master Theorem imposes far fewer restrictions on the shape of the recursive call, allowing multiple terms with floors, ceilings, and other deviations.

Apart from the more general recursion scheme that our version of the Master Theorem allows, there are two more differences to the version of the Master Theorem given by Cormen et al. [7]:

The second case of our version of the Master Theorem is more general, as it allows arbitrary real numbers q whereas the version by Cormen et al. require $q \geq 0$.[6]

The third case is more restrictive than that by Cormen et al.: we demand $g \in \Theta(x^q)$, whereas Cormen et al. only demand $g \in \Omega(x^q)$ and the existence of some $c < 1$ such that for all sufficiently large x, the regularity condition $a \cdot g(x/b) \leq c \cdot g(x)$ holds. For a more complex recursion scheme, as allowed by our Master Theorem, this regularity condition would be so complicated that it would be very inconvenient:

$$\sum_{i=1}^{k} a_i \cdot g(b_i \cdot x + h_i(x)) \leq c \cdot g(x)$$

Given such a regularity condition, the proof that f is then in $\Theta(g)$ is relatively simple and does not require the Akra–Bazzi theorem at all, which is why we did not include this in our proof.

It should also be noted that while most informal proofs of the Master Theorem mention that rounding before the recursive call does not change the result, this is seldom proven concisely. Cormen et al. give a partial proof that their Master Theorem also holds for $f(x) = a \cdot f(\lfloor x/b \rfloor) + g(x)$ and $f(x) = a \cdot f(\lceil x/b \rceil) + g(x)$, but they do not address the case $f(x) = a_1 \cdot f(\lfloor x/b \rfloor) + a_2 \cdot f(\lceil x/b \rceil) + g(x)$. This is unfortunate, because even simple Divide and Conquer algorithms such as Merge Sort have cost recurrences of this kind. Ad-hoc arguments using monotonicity can be made for concrete examples such as Merge Sort, but it is convenient to handle all of these deviations with a unified theorem like the Akra–Bazzi theorem.

[6] The Master Theorem presented in the book actually demands $q = 0$, but Exercise 4.2–2 is a generalisation to $q \geq 0$.

10 Conclusion

We formally verified a very general version of the Akra–Bazzi method [1] with the theorem prover Isabelle/HOL. This enables users of Isabelle to obtain verified asymptotic bounds for many typical ‘Divide and Conquer’ recurrence relations. In the process of our formalisation, we found a missing case in Leighton’s original proof [14]. We also clarified some important parts of the proof that Leighton does not address (such as the asymptotic inequalities and lifting the estimate for continuous recurrences to discrete ones) and slightly generalised the theorem.

Based upon our formal proof of the Akra–Bazzi method, we also proved a generalisation of the well-known Master Theorem whose generality is somewhere between the ‘classic’ Master Theorem and the Akra–Bazzi method, but with no additional cost compared to the ‘classic’ Master Theorem. In particular, we thus accounted rigorously for any rounding in the recursive calls, which is often neglected in informal proofs of the Master Theorem.

Additionally, we developed some automated proof methods that facilitate defining such recursive functions and working with our Master Theorem. We evaluated this machinery on some standard textbook examples of Divide and Conquer algorithms (Merge Sort, Karatsuba multiplication, deterministic Median-of-Medians selection) and found that the complexity proofs were almost completely automatic.

Acknowledgements Tobias Nipkow and Johannes Hölzl commented on an early draft of this work. Louay Bazzi made a very helpful suggestion that allowed us to fix the missing case in Leighton’s proof. We also thank the anonymous reviewers for their helpful suggestions and insightful questions.

Appendix: The Class of Admissible Integrals

We will now provide a proof of our claim from Sect. 3.3 that any admissible integral must coincide with the Riemann integral for all piecewise-continuous functions. First of all, we shall state again what admissibility means formally: Consider an integral notion $\mathcal{I}$. Formally, $\mathcal{I}$ consists of

– A functional $\mathcal{I} : \mathbb{R}^{\mathbb{R}} \times \mathbb{R}^2 \to \mathbb{R}$ that maps a function f and interval bounds a and b to the $\mathcal{I}$-integral of f from a to b, denoted by

$$\underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x\,.$$

– A set $\mathcal{I} \subseteq \mathbb{R}^{\mathbb{R}} \times \mathbb{R}^2$, the set of $\mathcal{I}$-integrable functions. We say that f is $\mathcal{I}$-integrable on $[a; b]$ if $(f, a, b) \in \mathcal{I}$.

For the remainder of this section, we will always indicate for every use of the integral sign which notion of integration is meant by writing the name of the integral underneath the integral as we did above.

Now assume that $\mathcal{I}$ fulfils the following four properties (which are the same that were stated in Sect. 3.3, but more formally):

$$\forall a, b, c \in \mathbb{R}.\quad a \le b \;\wedge\; c \ge 0 \;\longrightarrow\; ((\lambda x.\, c), a, b) \in \mathcal{I} \;\wedge\; \underset{\mathcal{I}}{\int_a^b} c\mathrm{d}x = c \cdot (a - b) \tag{9}$$

$$\forall a, b, a', b' \in \mathbb{R}.\, \forall f \in \mathbb{R}^{\mathbb{R}}.\quad a \le a' \le b' \le b \;\wedge\; (f, a, b) \in \mathcal{I} \;\longrightarrow\; (f, a', b') \in \mathcal{I} \tag{10}$$

$$\forall a, b \in \mathbb{R}. \forall f, g \in \mathbb{R}^{\mathbb{R}}.\quad (f, a, b) \in \mathcal{I} \ \wedge\ (g, a, b) \in \mathcal{I} \ \wedge\ (\forall x \in [a; b].\ f(x) \le g(x))$$
$$\longrightarrow \underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x \le \underset{\mathcal{I}}{\int_a^b} g(x)\,\mathrm{d}x \tag{11}$$
$$\forall a, b, c \in \mathbb{R}. \forall f \in \mathbb{R}^{\mathbb{R}}.\quad a \le b \le c \ \wedge\ (f, a, c) \in \mathcal{I}$$
$$\longrightarrow \underset{\mathcal{I}}{\int_a^c} f(x)\,\mathrm{d}x \ = \underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x + \underset{\mathcal{I}}{\int_b^c} f(x)\,\mathrm{d}x \tag{12}$$

To show that the $\mathcal{I}$-integral of a non-negative function f over $[a; b]$ coincides with the Riemann integral if f is both $\mathcal{I}$-integrable and piecewise-continuous on $[a; b]$, we recall that the Riemann integral is equivalent to the Darboux integral. The Darboux integral is defined as the supremum L_f of the lower Darboux sums $L_{f,P}$ and the infimum U_f of the upper Darboux sums $U_{f,P}$ over all subdivisions $P = (c_0 \ldots c_n)$ of the interval $[a; b]$ whenever L_f and U_f are equal. Formally:

$$\underset{\text{Darboux}}{\int_a^b} f(x)\,\mathrm{d}x := L_f = U_f \qquad (\text{if } L_f = U_f)$$

where

$$L_f = \sup_P\ L_{f,P} = \sup_P \sum_{i=0}^{n-1} (c_{i+1} - c_i) \inf_{x \in [c_i\,;\,c_{i+1}]} f(x) \quad \text{and} \quad U_f = \inf_P\ U_{f,P}$$
$$= \inf_P \sum_{i=0}^{n-1} (c_{i+1} - c_i) \sup_{x \in [c_i\,;\,c_{i+1}]} f(x)$$

Suppose f is non-negative, continuous, and $\mathcal{I}$-integrable on $[a; b]$. For any subdivision $P = (c_0 \ldots c_n)$ of the interval $[a; b]$, we can use (11) and (12) to split up the $\mathcal{I}$-integral over $[a; b]$:

$$\underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} \underset{\mathcal{I}}{\int_{c_i}^{c_{i+1}}} f(x)\,\mathrm{d}x$$

Using this together with the monotonicity and constant-interval property of $\mathcal{I}$, we have:

$$L_{f,P} \overset{\text{def}}{=} \sum_{i=0}^{n-1} (c_{i+1} - c_i) \inf_{x \in [c_i\,;\,c_{i+1}]} f(x) \overset{(9)}{=} \sum_{i=0}^{n-1} \underset{\mathcal{I}}{\int_{c_i}^{c_{i+1}}} \inf_{x \in [c_i\,;\,c_{i+1}]} f(x)\,\mathrm{d}x \overset{(11)}{\le} \underbrace{\sum_{i=0}^{n-1} \underset{\mathcal{I}}{\int_{c_i}^{c_{i+1}}} f(x)\,\mathrm{d}x}_{= \underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x} \overset{(11)}{\le}$$
$$\le \sum_{i=0}^{n-1} \underset{\mathcal{I}}{\int_{c_i}^{c_{i+1}}} \sup_{x \in [c_i\,;\,c_{i+1}]} f(x) \overset{(9)}{=} \sum_{i=0}^{n-1} (c_{i+1} - c_i) \sup_{x \in [c_i\,;\,c_{i+1}]} f(x) \overset{\text{def}}{=} U_{f,P}$$

Therefore, the $\mathcal{I}$-integral lies between all lower and upper Darboux sums. Since f is continuous, f is also Darboux-integrable, and therefore the supremum of the lower Darboux sums and the infimum of the upper Darboux sums are the same. Since the $\mathcal{I}$-integral lies inbetween, we have:

$$\underset{\text{Darboux}}{\int_a^b} f(x)\,\mathrm{d}x \overset{\text{def}}{=} L_f = U_f = \sum_{i=0}^{n-1} \underset{\mathcal{I}}{\int_{c_i}^{c_{i+1}}} f(x)\,\mathrm{d}x = \underset{\mathcal{I}}{\int_a^b} f(x)\,\mathrm{d}x.$$

We have therefore shown that $\mathcal{I}$ coincides with the Riemann integral on all continuous non-negative functions. Since we can split the $\mathcal{I}$-integral of a piecewise-continuous function into a sum of $\mathcal{I}$-integrals of continuous functions, this extends to all piecewise-continuous non-negative functions.

However, this result does not extend to more general notions of integrals and integrability: For example, let us consider the following integral $\mathcal{I}$: A function is $\mathcal{I}$-integrable if it is a constant function or if it is $[\mathbb{Q}] = (\lambda x.\ \text{if } x \in \mathbb{Q} \text{ then } 1 \text{ else } 0)$, the indicator function of the rational numbers. The value of the integral is defined as

$$\underset{\mathcal{I}}{\int_a^b} c\,\mathrm{d}x := c \cdot (b-a) \qquad \text{and} \qquad \underset{\mathcal{I}}{\int_a^b} [\mathbb{Q}]\,\mathrm{d}x := b-a$$

Then $\mathcal{I}$ fulfils all four properties, but unlike the Lebesgue/Bochner/Henstock–Kurzweil integral, the $\mathcal{I}$-integral of $[\mathbb{Q}]$ is non-zero on all non-empty intervals.

References

1. Akra, M., Bazzi, L.: On the solution of linear recurrence equations. Comput. Optim. Appl. **10**(2), 195–210 (1998). doi:10.1023/A:1018373005182
2. Avigad, J., Donnelly, K.: Formalizing O notation in Isabelle/HOL. In: Basin, D., Rusinowitch, M. (eds.) Automated Reasoning, *Lecture Notes in Computer Science*, pp. 357–371. Springer, Berlin (2004). doi:10.1007/978-3-540-25984-8_27
3. Avigad, J., Hölzl, J., Serafin, L.: A formally verified proof of the central limit theorem. CoRR **abs/1405.7012** (2014). Presented at the Isabelle Workshop 2014
4. Ballarin, C.: Locales: a module system for mathematical theories. J. Autom. Reason. **52**(2), 123–153 (2014). doi:10.1007/s10817-013-9284-7
5. Bazzi, L., Mitter, S.K.: The solution of linear probabilistic recurrence relations. Algorithmica **36**(1), 41–57 (2003). doi:10.1007/s00453-002-1003-4
6. Boncelet Jr., C.G.: Block arithmetic coding for source compression. IEEE Trans. Inf. Theory **39**(5), 1546–1554 (1993). doi:10.1109/18.259639
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 4th printing, 3rd edn. The MIT Press, Cambridge (2009)
8. Drmota, M., Szpankowski, W.: A Master theorem for discrete divide and conquer recurrences. J. ACM **60**(3), 16:1–16:49 (2013). doi:10.1145/2487241.2487242
9. Eberl, M.: The Akra–Bazzi Theorem and the Master Theorem. Archive of Formal Proofs (2015). http://www.isa-afp.org/entries/Akra_Bazzi.shtml, Formal proof development
10. Eberl, M.: Landau Symbols. Archive of Formal Proofs (2015). http://www.isa-afp.org/entries/Landau_Symbols.shtml, Formal proof development
11. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) Functional and Logic Programming. *Lecture Notes in Computer Science*, vol. 6009, pp. 103–117. Springer, Berlin (2010). doi:10.1007/978-3-642-12251-4_9
12. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: Reis, G.D., Théry, L. (eds.) Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS'09), pp. 38–45. Munich (2009)
13. Krauss, A.: Automating Recursive Definitions and Termination Proofs in Higher-Order Logic. Ph.D. thesis, Technische Universität München, Institut für Informatik (2009). http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20090722-681651-1-1
14. Leighton, T.: Notes on Better Master Theorems for Divide-and-Conquer Recurrences (1996). http://courses.csail.mit.edu/6.046/spring04/handouts/akrabazzi.pdf

C Analytic Number Theory

This chapter was originally published as an article in the proceedings of a peer-reviewed conference:

> M. Eberl.
> Nine Chapters of Analytic Number Theory in Isabelle/HOL.
> In *Interactive Theorem Proving*. Leibniz International Proceedings in Informatics, 2019.
> DOI: `10.4230/LIPIcs.ITP.2019.16`.

I am the sole author of this article, thus all contributions are mine.

Synopsis: This work consists of a formalisation of many basic concepts from the mathematical field of *Analytic Number Theory* (including Dirichlet series, character theory, and the Riemann ζ function). It then uses this as a basis for the formal proofs of some important theorems, such as the *Prime Number Theorem* and *Dirichlet's Theorem*. It covers most of Apostol's classic textbook *Analytic Number Theory* [3], sometimes going beyond the scope of the book.

Nine Chapters of Analytic Number Theory in Isabelle/HOL

Manuel Eberl
Technical University of Munich, Boltzmannstraße 3, Garching bei München, Germany
`https://www21.in.tum.de/~eberlm`
manuel.eberl@tum.de

Abstract

In this paper, I present a formalisation of a large portion of Apostol's *Introduction to Analytic Number Theory* in Isabelle/HOL. Of the 14 chapters in the book, the content of 9 has been mostly formalised, while the content of 3 others was already mostly available in Isabelle before.

The most interesting results that were formalised are:

- The Riemann and Hurwitz ζ functions and the Dirichlet L functions
- Dirichlet's theorem on primes in arithmetic progressions
- An analytic proof of the Prime Number Theorem
- The asymptotics of arithmetical functions such as the prime ω function, the divisor count $\sigma_0(n)$, and Euler's totient function $\varphi(n)$

2012 ACM Subject Classification Mathematics of computing → Mathematical analysis

Keywords and phrases Isabelle, theorem proving, analytic number theory, number theory, arithmetical function, Dirichlet series, prime number theorem, Dirichlet's theorem, zeta function, L functions

Digital Object Identifier 10.4230/LIPIcs.ITP.2019.16

Supplement Material The proof developments in the *Archive of Formal Proofs* (AFP) that this work refers to are listed in the bibliography. Additionally, a precise overview of what material from the book has been formalised and which theorems in the book correspond to which theorems in the formalisation can be found at 10.5281/zenodo.3262266.

Funding This work was supported by DFG grant NI 491/16-1. Part of it was conducted during a research visit in collaboration with the ALEXANDRIA project (ERC grant 742178).

Acknowledgements I would like to thank John Harrison for doing all the incredibly hard work of creating an extensive library of complex analysis in HOL Light – the first of its kind – and Larry Paulson and Wenda Li for porting it to Isabelle/HOL and extending it even further. Without these efforts, my work would not have been possible. Larry Paulson also started off the analytic proof of the Prime Number Theorem in Isabelle and allowed me to take over and replace it with a more high-level approach. I also thank Jeremy Avigad and Johannes Hölzl, who commented on a draft of this document, and the anonymous reviewers, who gave a number of helpful suggestions.

1 Introduction

The formalisation of Apostol's book in Isabelle/HOL started from the simple desire to have more properties about Euler's φ function available in the system. However, Apostol's style turned out to be very amenable to formalisation, and the subject matter was both of great interest as a basis for further development of number theory in Isabelle and as a case study for Isabelle's libraries on asymptotics and complex analysis. After 1.5 years of a highly part-time one-person effort, most of the book (and quite a bit of material that goes beyond the book) has been formalised:

- Chapters 1, 5, and 9 consist of fairly basic material (e. g. GCDs, congruences, Quadratic reciprocity), most of which was already available in the Isabelle/HOL library.
- The results from Chapters 2, 3, 4, and 6 have been formalised in their entirety.
- Chapters 10, 11, and 12 have been formalised with some omissions.
- For Chapters 7 and 13 (Dirichlet's Theorem and PNT), equivalent results have been formalised using a different approach.
- Chapters 8 and 14 have been skipped. The former is being actively worked on; the latter concerns number partitions and has little connection to the other material in the book.
- Various interesting results from other sources (e. g. Hildebrand's lecture notes [21]) that are not proven in Apostol's book have also been formalised.

For more precise information on this, see the supplementary material listed before.

I put particular focus on developing a usable library of Dirichlet series on one hand and concrete results about the distribution of primes on the other. As the development is much too large to be presented here in full, I will go through a high-level description of some of the most interesting material. Special attention will be given to parts where I encountered difficulties or chose a different route than Apostol did in his book. Proofs will only be given in the form of very brief sketches, e. g. when it is necessary in order to understand difficulties in formalising them. I would like to refer readers who are interested in the actual *proofs* either to my commented formalisation in the *Archive of Formal Proofs* (AFP) [13, 12, 15, 18, 16] or to the numerous excellent textbooks and lecture notes on the subject [2, 21, 7]. I chose not to show Isabelle code in this presentation since the main results are very close to mathematical notation (e. g. $\text{Re } s \geq 1 \Longrightarrow s \neq 1 \Longrightarrow \text{zeta } s \neq 0$) and showing the Isabelle code would therefore not provide much additional insight.

Let me now give an outline of the sections to follow: Section 2 lists related work. Section 3 defines formal Dirichlet series and their connection to complex-analytic functions. Section 4 introduces multiplicative characters and Dirichlet characters. Section 5 builds on the Dirichlet series library to treat various L functions, such as the famous Riemann ζ function. Section 6 describes my formalisation of the Prime Number Theorem (PNT). Section 7 gives some more examples of interesting results that were formalised. Section 8 gives an overview of the size of various parts of my formalisation and the effort involved in creating it. Lastly, in Section 9, some conclusions are drawn from this project.

▶ Remark 1. Any sum $\sum_p$ or product $\prod_p$ is to be understood to run over prime p only.

2 Related Work

The first formalisation of a result related to this work was that of the PNT in Isabelle/HOL by Avigad *et al.* [5] in 2007. They formalised the elementary Selberg–Erdős proof.[1] Carneiro formalised the same proof in Metamath [11].

Harrison developed the first (and until now only) formalisation of an analytic proof of the PNT in 3,600 lines [20] of HOL Light. He followed Newman's presentation, which I also did.[2]

[1] Unfortunately, this work was never submitted to the AFP and has not been maintained since then. At the time of writing this paper, the proofs are 12 years old; the formalisation comprises almost 27,000 lines, and many of them are unstructured proof scripts. Bringing them up to date to work with a modern version of Isabelle would be a massive undertaking. However, much of the more general material developed by Avigad *et al.* was moved to Isabelle's library, and for a considerable part of the remaining material, equivalent results are now already a part of the Isabelle library or my work anyway.

[2] Paulson later ported Harrison's development to Isabelle/HOL, but the ported proofs were lengthy and not very readable, so he and I decided that it would be better to redo them in a more high-level style, which I did. Only a few small lemmas were kept.

Harrison also proved Dirichlet's Theorem [19] and I used some of the high-level structure of his development as an inspiration for mine. Moreover, formalisations of Bertrand's postulate exist by Harrison in HOL Light, by Théry in Coq [27], by Riccardi in Mizar [26], by Carneiro in Metamath [10], by Asperti and Ricciotti in Matita [4], and by Biendarra and Eberl in Isabelle/HOL [9].

The big difference between these formalisations and the present one is that this one contains not just *one* result and the material required for it, but the majority of a textbook on the subject. Many proofs are much simpler and more "high-level" through the use of *Dirichlet series* and Isabelle's advanced machinery for asymptotic reasoning.

3 Dirichlet Series

The central objects in analytic number theory are *Dirichlet series*. These are the main tools that set apart my approach to formalised number theory from that of previous formalisation work in multiplicative number theory like that by Avigad *et al.*, Harrison, and Carneiro.

▶ **Definition 2** (Formal Dirichlet series). *A formal series of the form $f(s) = \sum_{n=1}^{\infty} a_n n^{-s}$ is called a Dirichlet series. Typically, the a_n are real or complex (we will mostly look at the complex case). The Dirichlet series over $\mathbb{R}$ or $\mathbb{C}$ form a commutative ring with the obvious choices for 0, 1, and addition. Multiplication is defined as $(\sum_{n=1}^{\infty} a_n n^{-s}) \cdot (\sum_{n=1}^{\infty} b_n n^{-s}) = \sum_{n=1}^{\infty} (\sum_{k \cdot l = n} a_k b_l) n^{-s}$. Also, $\sum_{n=1}^{\infty} a_n n^{-s}$ has a multiplicative inverse iff $a_1 \neq 0$.*

▶ **Theorem 3** (Convergence of Dirichlet series). *Each Dirichlet series has abscissæ of convergence σ_c and absolute convergence σ_a such that the infinite sum corresponding to it is absolutely summable for $\mathrm{Re}(s) > \sigma_a$, conditionally summable for $\mathrm{Re}(s) \in (\sigma_c, \sigma_a)$, and divergent for $\mathrm{Re}(s) < \sigma_c$ (where $\mathrm{Re}(s)$ denotes the real part of s). The σ_c and σ_a satisfy $\sigma_c \leq \sigma_a \leq \sigma_c + 1$ and may be $\pm\infty$.*

Much like formal power series (i. e. ordinary generating functions) for combinatorics, Dirichlet series are closely associated with number theory. Like generating functions, they are of great interest as mere formal objects, but when they converge, their interpretation as a complex-valued function is also enormously useful, as we will see.

Various formal analogues of analytic operations can be defined for Dirichlet series e. g. reciprocal, derivative, integral, $\exp(f(s))$, $\ln f(s)$, $f(s + s_0)$, $f(m \cdot s)$, and subseries. These have similar properties to their analytic counterparts (e. g. $\exp(f(s))' = f'(s) \exp(f(s))$) even when they do not converge. When they do converge, they typically agree with their analytic counterparts. This allows one to prove properties of the analytic functions by reasoning on the formal level and vice versa.

There are 4,800 lines of material on formal Dirichlet series in my formalisation. This is far too much to show here, so I simply say that it contains all of Chapter 11 in Apostol's book and more, except for Sections 11.10 and 11.11. I will only show a few small examples that illustrate the aforementioned interplay of the formal and the analytical level:

One example of using formal Dirichlet series to derive an analytic result is this:

▶ **Theorem 4.** *Let $\omega(n)$ be the number of distinct prime factors of n and $\mu(n)$ the Möbius μ function, i. e. $(-1)^{\omega(n)}$ if n is square-free and 0 otherwise. Then $\sum_{n=1}^{\infty} \mu(n)/n^2 = 6/\pi^2$.*

Proof. Consider the formal series $\zeta(s) := \sum_{n=1}^{\infty} n^{-s}$ and $M(s) := \sum_{n=1}^{\infty} \mu(n) n^{-s}$. It is clear that they both converge absolutely for $\mathrm{Re}(s) > 1$ by the comparison test. It is easy to show $\sum_{d|n} \mu(d) = [n = 1]$, i. e. $\zeta(s)M(s) = 1$ holds formally [2]. Thus, it also holds analytically for $\mathrm{Re}(s) > 1$ so that we have $\zeta(2)M(2) = 1$ and therefore $M(2) = 1/\zeta(2)$, where $\zeta(2)$ – the famous *Basel problem* – has the well-known value $\pi^2/6$ [6]. ◀

The following theorem allows us to transfer an analytic equality to the formal level:

▶ **Theorem 5** (Uniqueness of Dirichlet series). *Let $f(s)$, $g(s)$ be two formal Dirichlet series whose abscissa of convergence is $< \infty$. If there exists a sequence s_k with $\mathrm{Re}(s_k) \longrightarrow \infty$ and $\forall k.\ f(s_k) = g(s_k)$, then $f(s)$ and $g(s)$ are equal as formal Dirichlet series.*

▶ Remark 6. In Isabelle, the condition on the existence of the sequence s_k is replaced by the following equivalent and more concise formulation using *filters* [22]:

$$\exists_F\ s\ \textbf{in}\ \mathrm{Re}\ \textbf{going-to}\ \text{at-top}.\ f(s) = g(s)$$

The filter "f **going-to** F" is the contravariant image of F under f, i. e. "Re **going-to** at-top" describes the neighbourhood of complex numbers with "sufficiently large" real part.

The "$\exists_F\ x$ **in** F. $P(x)$" notation stands for "$P(x)$ holds frequently in F", i. e. the complement of P is not in the filter F. Less formally, one could say that $P(x)$ holds "again and again". In the case of "Re **going-to** at-top", this means that for any $C \in \mathbb{R}$, there exists an s with $\mathrm{Re}(s) \geq C$ for which the property is fulfilled.

▶ **Definition 7** (Truncation operator). *For a Dirichlet series $f(s) = \sum_{n=1}^{\infty} a_n n^{-s}$, let $T_m(f(s)) = \sum_{n=1}^{m} a_n n^{-s}$ denote the m-th order truncation of $f(s)$. The result is a Dirichlet polynomial, i. e. a Dirichlet series with only finitely many non-zero coefficients.*

▶ **Theorem 8.** $f(s) = g(s) \longleftrightarrow \forall m.\ T_m(f(s)) = T_m(g(s))$.

The following is an instance where these theorems are used to avoid a lot of complicated reasoning on the formal level by leveraging a result on the analytic level:

▶ **Theorem 9.** *For any (not necessarily convergent) Dirichlet series $f(s)$ and $g(s)$, we have* $\exp(f(s) + g(s)) = \exp(f(s)) \exp(g(s))$.

Proof. It is clear that the result holds analytically whenever the series converge, so if the series have a non-empty half-plane of convergence, they must be equal by Theorem 5. The question is how to show this if we do *not* know whether the series converge anywhere.

The key is to use Theorem 8 together with $T_m(\exp(h(s))) = T_m(\exp(T_m(h(s))))$. This allows us to assume w. l. o. g. that the series in question are Dirichlet polynomials and therefore converge everywhere. ◀

▶ Remark 10. This technique of showing an equality on Dirichlet series by showing that it holds for all Dirichlet polynomials works if the two sides of the equation in question are continuous functions w. r. t. the topology on formal Dirichlet series, i. e. each coefficient of the result only depends on finitely many coefficients of the input. The topological structure of Dirichlet series was not formalised yet, but this is a useful fact to keep in mind.

The following is another important theorem connecting a series with the function it defines that we will use later:

▶ **Theorem 11** (Pringsheim–Landau). *Let $f(s)$ be a Dirichlet series with non-negative real coefficients and $\sigma_a \neq \pm\infty$. Then $f(s)$ has a singularity at σ_a.*

Conversely, if $f(s)$ has an an analytic continuation to some half-plane $\{s \mid \mathrm{Re}(s) > c\}$, then $\sigma_a \leq c$. In particular, if $f(s)$ is entire, the series must converge everywhere.

4 Characters of a Finite Abelian Group

The next concept we shall explore is that of a *multiplicative character*, which will be needed to prove Dirichlet's theorem. For this section, let $G = (G, \cdot, 1)$ be a finite abelian group.

▶ **Definition 12** (Multiplicative character). *A character is a group homomorphism $\chi : G \to \mathbb{C}^\times$, i. e. $\chi(1) = 1$ and $\chi(a \cdot b) = \chi(a)\chi(b)$ for any $a, b \in G$. The character χ_0 that maps every element to 1 is called the principal character.*

For the necessary group theory, I use the `HOL-Algebra` library by Ballarin, which models a group as a record containing entries for the operation $\cdot$, the neutral element 1, and an explicit carrier set (which does not have to be the full type universe). The latter is necessary in HOL because notions such as subgroups cannot easily be expressed without explicit carrier sets. The fact that such a record indeed describes a group is then formalised as a *locale* [8] called *group*, which fixes such a record and assumes that all the usual group axioms hold.

A character can then be defined as a locale that extends the *group* locale by fixing a function $\chi :: \alpha \to \mathbb{C}$ (where α is the type of the group elements) and assuming that the two homomorphism properties mentioned above hold for χ. For convenience, I only assume $\chi(1) \neq 0$ (from which $\chi(1) = 1$ easily follows) and I additionally require $\chi(x) = 0$ for any x not in the carrier of the group. The latter is to ensure *extensionality*, i. e. two characters are equal as HOL values iff they return the same result on every group element.

▶ **Definition 13** (Pontryagin dual group). *Denote the set of characters of G by $\widehat{G}$. Then $\widehat{G}$ forms a group $\widehat{G} := (\widehat{G}, \cdot, \chi_0)$ with point-wise multiplication and χ_0 as the identity. This group is called the Pontryagin dual group of G.*

▶ **Theorem 14** (Number of characters). $|\widehat{G}| = |G|$

In Isabelle, the proof is by induction on the subgroups of G, using a custom induction rule inspired by Apostol's proof. The idea here is to successively "adjoin" elements, i. e. for a subgroup H and some $x \in G \setminus H$, we form $\langle H; x\rangle$, the subgroup generated by $H \cup \{x\}$:

▶ **Lemma 15** (Induction on a group). *Let $G = (G, \cdot, 1)$ be a group and H some subgroup of G. Let P be some property on groups. If $P(H)$ holds and $P(H')$ implies $P(\langle H'; x\rangle)$ for all subgroups $H' \supseteq H$ and all $x \in G \setminus H'$, then $P(G)$ holds.*

I use this to show a stronger version of Theorem 14 that is just as easy to show:

▶ **Theorem 16** (Number of character extensions). *Let H be a subgroup of G and $\chi \in \widehat{H}$. Let*

$$C(G) := \{\chi' \in \widehat{G} \mid \forall x \in H.\ \chi'(x) = \chi(x)\}$$

denote the set of characters on G that agree with χ on H. Then $|C(G)| \cdot |H| = |G|$, i. e. there are precisely $|G|/|H|$ ways to extend a character on H to a character on G.

Proof. By straightforward induction according to Lemma 15, using the bijection

$$f : C(\langle H'; x\rangle) \to C(H') \times \{z \in \mathbb{C} \mid z^n = 1\}, \quad f(\chi) = (y \mapsto \chi(y), \chi(x))$$

in the induction step. ◀

Theorem 14 follows directly by taking $H = (\{1\}, \cdot, 1)$. Another useful corollary is this:

▶ **Corollary 17.** *For any $x \neq 1$, there exists a $\chi \in \widehat{G}$ such that $\chi(x) \neq 1$.*

With this, we can prove a nice property that Apostol does not cover at all:

▶ **Theorem 18** (Isomorphism to the double dual). *G is isomorphic to its double dual via the natural isomorphism $\nu : G \to \widehat{\widehat{G}}\,,\ \nu(x) = (\chi \mapsto \chi(x))\,.$*

This isomorphism is useful for the next properties:

▶ **Theorem 19** (Orthogonality relations). *For any $\chi \in \widehat{G}$ resp. $x \in G$, we have:*

$$\sum_{x\in G} \chi(x) = \begin{cases} |G| & \text{if } \chi = \chi_0 \\ 0 & \text{otherwise} \end{cases} \qquad (1) \qquad\qquad \sum_{\chi\in \widehat{G}} \chi(x) = \begin{cases} |G| & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

Apostol's proof for (1) is very simple and straightforward to formalise. In order to show (2) from (1), Apostol represents the set of characters of G as a *character table* of G, i. e. a $|G| \times |G|$ complex matrix. If we denote this matrix by A, (1) shows that $AA^* = nI$ (where A^* is the conjugate transpose of A). By simple linear algebra, $A^*A = nI$ and thus (2) follows.

Formalising this argument would have required importing Isabelle's linear algebra library and doing some tedious work to relate the matrix to the characters, so I chose another route: One *could* prove (2) relatively easily using the same induction principle as before in about 70 lines, but the easiest way is to simply use Pontryagin duality: (2) is, in fact, nothing but the dual of (1), with $\widehat{G}$ for G and $\nu(x)$ for χ. This requires only 6 lines of Isabelle code.

▶ **Definition 20** (Dirichlet character). *A Dirichlet character χ for the modulus $m \in \mathbb{N}_{>1}$ is a character of the multiplicative group of the residue ring $\mathbb{Z}/m\mathbb{Z}$. For convenience, χ is represented as a periodic function of type $\mathbb{N} \to \mathbb{C}$ with period m, i. e. $\chi(k) = \chi(k \text{ mod } m)$.*

▶ Remark 21. Apostol's and my treatment of characters are quite elementary. There is an alternative, more group-theoretic view on this: It is straightforward to show that $\widehat{G_1 \times G_2} \simeq \widehat{G}_1 \times \widehat{G}_2$ and that $\widehat{C}_n \simeq C_n$ for cyclic groups C_n. Together with the *Fundamental Theorem of Finite Abelian Groups*, this implies a stronger variant of Theorem 14, namely $\widehat{G} \simeq G$. However, unlike with $\widehat{\widehat{G}} \simeq G$, the isomorphism is not natural and establishing it indeed requires the Fundamental Theorem, which is currently not available in `HOL-Algebra`. Since the formal proofs that were presented in this section are still reasonably short, I do not think this is a big problem.

5 The L Functions

In this section, we will look at four functions from the class of L functions: Riemann's ζ function, Dirichlet's L function, Hurwitz's ζ function, and the periodic ζ function. These are all complex-valued functions that are defined by an infinite sum for $\text{Re}(s) > 1$ and can be analytically or meromorphically continued to the entire complex plane.

▶ **Definition 22** (Riemann's ζ function). *For $\text{Re}(s) > 1$, the Riemann ζ function is given by the Dirichlet series $\zeta(s) = \sum_{n=1}^{\infty} n^{-s}$.*

▶ **Definition 23** (Dirichlet L functions). *Let χ be a Dirichlet character for the modulus $m > 0$. Then $L(s, \chi)$ is given by the Dirichlet series $L(s, \chi) = \sum_{n=1}^{\infty} \chi(n)n^{-s}$ for $\text{Re}(s) > 1$ if $\chi = \chi_0$ and for $\text{Re}(s) > 0$ if $\chi \neq \chi_0$.*

We immediately get the following properties for free from the Dirichlet series library:

▶ **Theorem 24.** *Let $\Lambda(n)$ denote the von Mangoldt function. Then, if* $\mathrm{Re}(s) > 1$*:*

$$\zeta(s) = \prod_p \frac{1}{1-p^{-s}} \qquad \zeta'(s) = -\sum_{n=1}^{\infty} \frac{\ln n}{n^s} \qquad \ln\zeta(s) = \sum_{n=2}^{\infty} \frac{\Lambda(n)}{\ln n \cdot n^s}$$

$$L(s,\chi) = \prod_p \frac{1}{1-\chi(p)p^{-s}} \qquad L'(s,\chi) = -\sum_{n=1}^{\infty} \frac{\chi(n)\ln n}{n^s} \qquad \ln L(s,\chi) = \sum_{n=2}^{\infty} \frac{\chi(n)\Lambda(n)}{\ln n \cdot n^s}$$

However, $\zeta(s)$ and $L(s,\chi)$ can be defined on a larger domain:

▶ **Theorem 25** (Analytic continuation of $\zeta(s)$ and $L(s,\chi)$)**.**
1. *$\zeta(s)$ can be continued to an analytic function on $\mathbb{C}\setminus\{1\}$ with a simple pole at $s=1$.*
2. *For non-principal χ, $L(s,\chi)$ can be continued to an entire function.*
3. *For $\chi=\chi_0$, we have $L(s,\chi_0)=\zeta(s)\cdot\prod_{p|m}(1-p^{-s})$, i. e. $L(s,\chi_0)$ is equal to $\zeta(s)$ up to an entire factor and is therefore also analytic on $\mathbb{C}\setminus\{1\}$ with a simple pole at $s=1$.*

The difficult part here is to actually construct the analytic continuations. To do this uniformly and without duplication of work, Newman uses a generalisation of $\zeta(s)$:

▶ **Definition 26** (Hurwitz's ζ function)**.** *Let $a\in\mathbb{R}_{>0}$ and $\mathrm{Re}(s)>1$. Then $\zeta(s,a)$ is given by the (non-Dirichlet) series $\zeta(s,a)=\sum_{n=0}^{\infty}(n+a)^{-s}$.*

Apostol only considers $\zeta(s,a)$ for $a\in(0,1]$ since some results only hold for $a\le 1$ and the case of $a>1$ can be reduced to $a\in(0,1]$. It is, however, useful to allow also $a\ge 1$ – e. g. in Newman's proof of the PNT, as noted already by Harrison [20].

▷ Claim 27. $\zeta(s,a)$ can be continued analytically to $\mathbb{C}\setminus\{1\}$ with a simple pole at $s=1$.
Both Riemann's ζ function and the Dirichlet L functions can easily be expressed in terms of $\zeta(s,a)$, so that a continuation for Hurwitz's ζ also yields continuations for the other two [2].

The main question now is therefore how to construct the continuation of $\zeta(s,a)$.

5.1 Analytic Continuation of Hurwitz's ζ Function

Apostol constructs the continuation using an integral along an infinite contour. I did formalise this eventually (see Theorem 36), but when I first defined $\zeta(s,a)$ in Isabelle, this approach seemed quite daunting to me, so I chose another route that seems to be folklore [2, 24] and that I discovered independently: Since $\zeta(s,a)$ is defined for $\mathrm{Re}(s)>1$ by an infinite sum $\sum_{n=0}^{\infty}(n+a)^{-s}$ and the corresponding improper integral $\int_0^{\infty}(x+a)^{-s}\,\mathrm{d}x$ is easy to compute, the Euler–MacLaurin summation formula [14] suggests itself. Applying it, we obtain

$$\sum_{n=0}^{\infty}(s+a)^{-n} - \frac{a^{1-s}}{s-1} = \frac{a^{-s}}{2} + \sum_{i=1}^{N}\frac{B_{2i}}{(2i)!}\,a^{-s-2i+1}s^{\overline{2i-1}} + \frac{(-1)^{2N}s^{\overline{2N+1}}}{(2N+1)!}\int_0^{\infty}P_{2N+1}(t)\cdot(t+a)^{-s-2N-1}\,\mathrm{d}t \tag{3}$$

where $s^{\overline{k}}$ denotes the rising factorial, B_k is the k-th Bernoulli number, and $P_k(t)$ is the periodic version of the Bernoulli polynomial $B_k(t)$, i. e. $P_k(t)=B_k(t-\lfloor t\rfloor)$.

The right-hand side is now actually analytic on a larger domain: all terms except the last one are clearly entire functions in s; the only non-obvious term is the integral in the last summand. Leibniz's rule shows that the definite integral $\int_0^b$ is analytic in s, and an integral version of the Weierstraß M-test then shows that the improper integral $\int_0^{\infty}$ is uniformly convergent and therefore analytic in s for $\mathrm{Re}(s)>-2N$.

Let us write *prezeta*$_N(s,a)$ for the right-hand side. This is then a function in s that is analytic for $\mathrm{Re}(s) > -2N$ and that also fulfils

$$\mathrm{prezeta}_N(s,a) = \sum_{n=0}^{\infty}(s+a)^{-n} - \frac{a^{1-s}}{s-1} \qquad \text{for } \mathrm{Re}(s) > 1\ .$$

This means that two functions *prezeta*$_M$ and *prezeta*$_N$ will always agree on $\mathrm{Re}(s) > 1$, and by analytic continuation they will then also agree on their entire domain, i. e. for all s with $\mathrm{Re}(s) > -2\max(M,N)$. We can therefore define a full analytic continuation to all of $\mathbb{C}$ by choosing N "big enough" for each input, i. e. we define:

$$\mathrm{prezeta}(s,a) := \mathrm{prezeta}_{\max(0,1-\lceil \mathrm{Re}(s)/2\rceil)}(s,a)$$

This function is entire and agrees with any of the *prezeta*$_N(s,a)$ for all s with $\mathrm{Re}(s) > -2N$. Thus, it is an analytic continuation of the left-hand side of (3) so that we can simply define

$$\zeta(s,a) := \mathrm{prezeta}(s,a) + \frac{a^{1-s}}{s-1}$$

to obtain the Hurwitz ζ function on all of $\mathbb{C}\setminus\{1\}$. For convenience, I choose $\zeta(1,a) = 0$ as is often done in HOL-based systems (cf. $\Gamma(-n)$ for $n \in \mathbb{N}$ in Isabelle/HOL and HOL Light). The advantage of the Euler–MacLaurin approach is that it is simple to implement because all of the "heavy lifting" has already been done in the AFP entry on the Euler–MacLaurin formula.

Various basic properties of the Hurwitz and Riemann ζ functions then follow in a straightforward way, of which I show some notable ones here:

▶ **Theorem 28** (Special values of ζ). *For any $n \in \mathbb{N}_{\geq 0}$, we have:*

$$\zeta(a,-n) = -\frac{B_{n+1}(a)}{n+1} \qquad \zeta(-n) = -\frac{B_{n+1}}{n+1} \qquad \zeta(2n) = \frac{(-1)^{n+1}\cdot B_{2n}\cdot(2\pi)^{2n}}{2(2n)!}$$

where $B_n = B_n(1)$ are the Bernoulli numbers with $B_1 = \frac{1}{2}$. In particular, this implies the famous $\zeta(-1) = -\frac{1}{12}$ and the Basel problem $\zeta(2) = \pi^2/6$.

▶ **Theorem 29** (Integral representation for $\zeta(s,a)$). *For any s with $\mathrm{Re}(s) > 1$, we have:*

$$\Gamma(s)\zeta(s,a) = \int_0^\infty \frac{t^{s-1}e^{-at}}{1-e^{-t}}\,dt$$

5.2 The Non-Vanishing of $\zeta(s)$ and $L(s,\chi)$ for $\mathrm{Re}(s) = 1$

The following is a core ingredient in the Prime Number Theorem and Dirichlet's Theorem:

▶ **Theorem 30.** *For any s with $\mathrm{Re}(s) \geq 1$, we have $\zeta(s) \neq 0$ and $L(s,\chi) \neq 0$.*

The case of $\mathrm{Re}(s) > 1$ is a simple consequence of the Euler product formula for $\zeta(s)$ and $L(s,\chi)$ (cf. Theorem 24); the difficult part is the case $\mathrm{Re}(s) = 1$. For this, I formalised the very simple proof presented by Newman [25], whose key ingredient is the aforementioned Pringsheim–Landau theorem (see Theorem 11). This proof is stunningly short and its high-level reasoning translates well to Isabelle/HOL now that a library of Dirichlet series is available. The gain is most striking for the Dirichlet L function, where Apostol's proof only treats the case of $s = 1$, and even that proof is still more complicated than Newman's and involves lengthy complicated "Big-O" reasoning. Indeed, in a first version of the formalisation, I formalised Apostol's proof, but it was considerably longer and messier than the new version – with the added bonus that the new one is also more general.

Harrison also only proves $L(1,\chi) \neq 0$ – indeed, he does not define $L(s,\chi)$ at all; he defines only $L(1,\chi)$ since that is all that is required for Dirichlet's theorem. Despite this and the much higher verbosity of structured Isabelle proofs compared to HOL Light,his proof is longer than mine. The reason for this is that his proof is very elementary and uses very little library material while mine builds on a large library of Dirichlet series. However, I think that the comparison is still not entirely unjustified since all of this material is sufficiently general to be called "library material" (as opposed to technical lemmas specifically designed for this one proof), and building sufficiently large and general libraries to make proofs like this cleaner and easier is, after all, one of our goals in formalisation.

5.3 Hurwitz's Formula

More as a challenge to myself and the Isabelle libraries, I chose to formalise another non-trivial property of the ζ functions:

▶ **Theorem 31** (Reflection formula for $\zeta(s)$). *For $s \notin \{0,1\}$, we have:*

$$\frac{1}{\Gamma(s)} \cdot \zeta(1-s) = 2(2\pi)^{-s}\cos(\pi s/2)\zeta(s)$$

Note that while $\Gamma(s)$ has poles at $s \in \mathbb{Z}_{\leq 0}$, its reciprocal $1/\Gamma(s)$ is entire, so the formula holds even for $s \in \mathbb{Z}_{<0}$.

This formula is a corollary of a more general one for $\zeta(s,a)$ known as *Hurwitz's formula*:

▶ **Theorem 32** (Hurwitz's formula). *Let $a \in (0,1)$ and $s \in \mathbb{C} \setminus \{0\}$ with $a \neq 1 \vee s \neq 1$. Then:*

$$\frac{1}{\Gamma(s)} \cdot \zeta(1-s,a) = (2\pi)^{-s}\big(i^{-s}F(s,a) + i^{s}F(s,-a)\big)$$

Here, $F(s,a)$ is the *periodic ζ function*, which we still have to define:

▶ **Definition 33** (Periodic ζ function). *For $\mathrm{Re}(s) > 1$, the periodic ζ function $F(s,a)$ is given by the Dirichlet series $F(s,a) = \sum_{n=1}^{\infty} e^{2i\pi na}n^{-s}$.*

▷ Claim 34. $F(s,a)$ is called *periodic* because $F(s,a+n) = F(s,a)$ for any integer n. For non-integer a, the above series converges for $\mathrm{Re}(s) > 0$ and can be continued to an entire function. For integer a, it is simply the Riemann ζ function.

Apostol does not discuss the analytic continuation of $F(s,a)$ at all, but it seemed useful to me to do this nonetheless. The strategy I used to construct the continuation of $F(s,a)$ for non-integer a is somewhat interesting: Theorem 32 can be rearranged to give a formula that expresses $F(s,a)$ in terms of $\zeta(1-s,a)$ and $\zeta(1-s,1-a)$:

▶ **Theorem 35.** *Let $a \in (0,1)$ and $s \in \mathbb{C} \setminus \mathbb{N}$. Then:*

$$F(s,a) = i(2\pi)^{s-1}\Gamma(1-s)\big(i^{-s}\zeta(1-s,a) - i^{s}\zeta(1-s,1-a)\big)$$

We therefore proceed like this (assuming w. l. o. g. $a \in (0,1)$):
1. Show Theorem 32 for $\mathrm{Re}(s) > 1$ (where F is simply given by its Dirichlet series).
2. Use this to show Theorem 35 for $\mathrm{Re}(s) > 1$.

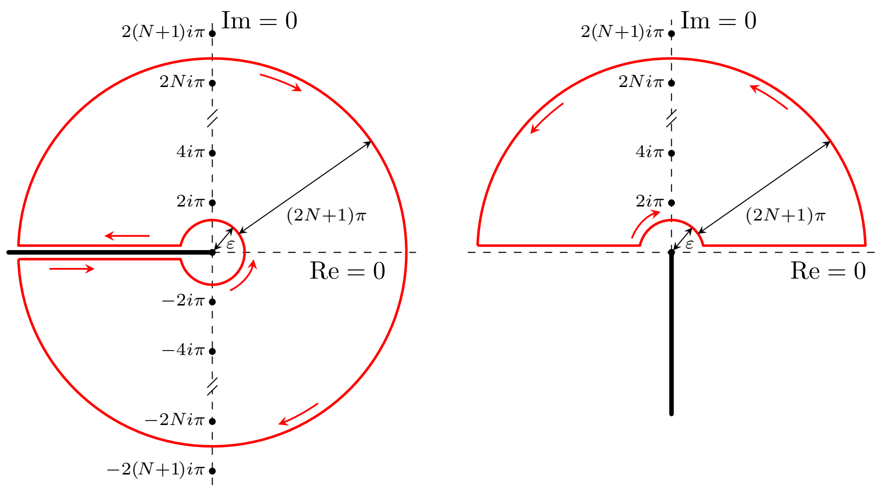


Figure 1 Apostol's integration contour and my modified version for proving Hurwitz's formula ($\varepsilon < 2\pi$). The black dots are the poles of the integrand; the thick black line is its branch cut. Note that in both cases, the line segments of the contour lie *on* the real axis despite the small gap in the illustration.

3. Use the right-hand side of Theorem 35 as the definition of $F(s, a)$ for $s \notin \mathbb{N}$. Compatibility with the Dirichlet series definition follows by analytic continuation.
4. Since the Dirichlet series definition covers $\mathrm{Re}(s) > 0$ and the new definition covers $\mathbb{C} \setminus \mathbb{N}$, the only point left is $s = 0$, which is a removable singularity that can be eliminated via

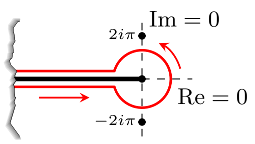
$$F(0, a) := \lim_{s\to 0} F(s, a) = \frac{i}{2\pi}\big(\mathrm{prezeta}(1, a) - \mathrm{prezeta}(1, 1 - a) + \ln(1 - q) - \ln q\big) - \frac{1}{2}\ .$$

5. Extend the validity of Theorems 32 and 35 to their full domains by analytic continuation.

The only difficult part here is the first step, which we shall look at now. First of all, we require the contour integral representation for $\zeta(s, a)$ mentioned in Section 5.1:

▶ **Theorem 36.** *For any $s \in \mathbb{C} \setminus \{1\}$, we have*

$$\zeta(s, a) = \frac{\Gamma(1 - s)}{2i\pi} \int_{\rightleftarrows\circ} \frac{z^{s-1} e^{az}}{1 - e^z}\, dz \qquad \textit{where } \rightleftarrows\circ =$$



if the inner circle has radius $\varepsilon < 2\pi$. This continues $\zeta(s, a)$ analytically to $\mathbb{C} \setminus \{1\}$.

Proof. Due to analytic continuation, we can assume $\mathrm{Re}(s) > 1$ w.l.o.g. By homotopy, all contours with a radius $\varepsilon < 2\pi$ yield the same result. Letting $\varepsilon \to 0$, the contribution of the circle vanishes and the $\int_{\rightleftarrows\circ}$ becomes the $\int_0^\infty$ from Theorem 29. ◀

▶ Remark 37. Note that in order to even state this theorem formally, one needs to make the limit inherent in this "improper contour integral" explicit. I chose to decompose the integral as $\int_{\rightleftarrows\circ} = \int_{\leftarrow\frown} - \int_{\rightarrow\smile}$. The two line segments can then be written as Lebesgue integrals $\int_0^{-\infty}$, leaving only two finite circular arcs as the remainder. There is yet another subtlety hidden in this integral that will be discussed in Section 5.3.1.

The proof of Hurwitz's formula for $\text{Re}(s) > 1$ then proceeds by computing this contour integral in a different way using the Residue Theorem. To do this, we first need to approximate it by an integral over a *finite* contour $C_{N,\varepsilon}$ such that $\int_{C_{N,\varepsilon}} \longrightarrow \int$ as $N \to \infty$. Figure 1 shows Apostol's choice for $C_{N,\varepsilon}$. Applying the Residue Theorem to this, we get

$$\frac{1}{2i\pi} \int_{C_{N,\varepsilon}} \frac{z^{-s} e^{az}}{1 - e^z} \, \mathrm{d}z = \sum_{z_0} \text{ind}_{C_{N,\varepsilon}}(z_0) \operatorname*{Res}_{z=z_0} \frac{z^{-s} e^{az}}{1 - e^z} \tag{4}$$

where the sum on the right-hand side extends over all the singularities of the integrand (represented by black dots in Figure 1). We can now let $N \to \infty$ so that the contribution of the outer circle vanishes. The integral on the left-hand side is then simply a $\int$, which is equal to $\zeta(s, a)/\Gamma(s)$ by Theorem 36, and winding number on the right-hand side -1 for every non-zero pole z_0. Evaluating this sum, we find that it indeed equals $\frac{i^{-s}}{(2\pi)^s} F(s, a) + \frac{i^s}{(2\pi)^s} F(s, -a)$, which concludes the proof of Hurwitz's formula for $\text{Re}(s) > 1$. ⌟

The formalisation of the proof was fairly routine. It is, however, quite large and tedious, containing almost 1,000 lines of proof code compared to 6.5 pages in Newman's book (both including the proof of Theorem 36). This seems to be a common pattern in Isabelle proofs using the Residue Theorem and it is likely due to the many side conditions that need to be shown, many of which are of geometric nature and thus much easier to explain to a human than to a theorem prover. Side conditions like the analyticity of the integrand, on the other hand, can be solved mostly automatically using Isabelle's general-purpose automation together with specialised theorem collections like `analytic_intros`.

Some aspects of the formal proofs of these statements deserve more attention, and we will discuss them now.

5.3.1 Branch Cuts

In both theorems, the term z^{-s} is a multi-valued function. It is defined in Isabelle as $e^{-s \ln z}$ where ln is the standard branch of the logarithm, which has a branch cut on the negative real axis. The two lines of Apostol's contour lie directly on this cut, taking different branches of the logarithm (indeed, if they did not, they would simply cancel each other). This makes sense formally when considering the integrand as a multi-valued function in the sense of a Riemann surface, but we do not have any of this analytic machinery in Isabelle.

My first idea to circumvent this problem was to resort to some kind of limiting argument by placing the two horizontal lines not directly on the real axis, but some ε above (resp. below) it. However, this would likely have been a very tedious argument to do in Isabelle. I therefore decided to again cut the contour into two halves, similarly to Remark 37. When their integrals are added together, we recover Apostol's contour integral. Due to symmetry, it is actually again enough to look at the upper half (cf. the right part of Figure 1), as the lower one follows by conjugation.

For this upper contour, we can now integrate over the same branch of the logarithm everywhere. In order to avoid the branch cut of the standard logarithm, I use a different branch $\widetilde{\ln}\, z := \ln(-iz) + \frac{1}{2} i\pi$, whose branch cut lies on the negative imaginary axis, safely away from our contour. I also reversed the contour so that the winding numbers are all 1.

5.3.2 Homotopy

The proof of Theorem 36 uses the fact that the integral along is invariant for all radii $\varepsilon < \pi$. This is because all of these contours are homotopic, i. e. they can be continuously deformed into one another without crossing any of the singularities of the integrand. However, proving that this is the case turned out to be very tedious in Isabelle because there are almost no library theorems that help showing that two composite paths are homotopic.

I circumvented this problem in the following way: First of all, I restricted myself to $\varepsilon < \pi$.[3] Next, since the line segments extending from $-\pi$ to $-\infty$ are the same for all ε, we can ignore them and focus on the finite subcontour $\multimap$. It can be seen that $\int_{\multimap} = \int_{\frown} - \int_{\smile}$.

By symmetry, it is enough to show that $\int_{\frown}$ is invariant under changes of ε. This, on the other hand, is actually a corollary of (4): If we let $N := 0$, the sum on the right-hand side vanishes so that we get $\int_{C_{0,\varepsilon}} = \int_{\Cap} = 0$ for all ε. Since $\int_{\frown} = \int_{\Cap} - \int_{\bigcap} = -\int_{\bigcap}$ and $\bigcap$ (a half circle of radius π) is independent of ε, it follows that $\int_{\frown}$ is indeed the same for all ε.

Effectively, this replaces the homotopy argument (which is intuitively obvious for humans and not mentioned at all by Apostol) with a much "heavier" invocation of the Residue Theorem – but since we already applied the Residue Theorem anyway, all that work is already done.

5.3.3 Winding Numbers

The evaluation of the winding numbers $\text{ind}_{C_{N,\varepsilon}}(z_0)$ is also easy for a human: the contour $C_{N,\varepsilon} = \Cap$ *clearly* winds counter-clockwise once around each pole $2ni\pi$ with $0 < n \leq N$, and all the other poles are *clearly* completely outside the contour. Proving these things in a theorem prover, on the other hand, is notoriously difficult [20], especially for a more complicated contour like this.

To show that the poles outside the contour really do lie outside (i. e. have winding number 0), I use simple geometric arguments: for the branch cut on the negative imaginary axis, one can draw a vertical line from each point to $-i\infty$ without crossing $C_{N,\varepsilon}$, so the winding number for these points must be 0. Moreover, $C_{N,\varepsilon}$ is contained in a ball of radius $(2N+1)\pi$, which is a convex set that does not contain any of the poles with $n > N$. Thus, these poles must also have winding number 0.

The more difficult part is to show that the winding number for the points inside the contour is 1. Geometric arguments for this are difficult. One approach would be to show that the contour is a closed simple curve (which implies that the winding number must be either -1, 0, or 1) and then weigh the contributions of the four different parts of the curve to show that the overall value must be positive, thus 1. However, to avoid having to do this work, I instead use Li's framework for computing winding numbers in Isabelle [23]. It is based on computing Cauchy indices and comes with some setup to handle combinations of line segments and circular arcs almost automatically, allowing me to prove that the winding numbers are 1 with a mere 18 lines of proof code.

6 The Prime Number Theorem

The formal statement of the PNT is simply the asymptotic estimate $\pi(x) \sim x \ln x$, where $\pi(x)$ is the number of prime numbers $\leq x$. I will now explain, in a high-level way, how the formalised proof works. First of all, let us define the following functions related to primes:

[3] This restriction could easily be lifted by allowing arbitrary radii in (4) instead of just $(2N+1)\pi$.

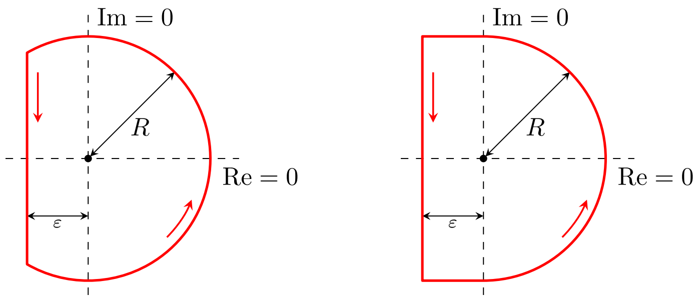


Figure 2 Newman's integration contour in his proof of Ingham's Tauberian theorem and Harrison's modified version. The dot in the middle is the pole of the integrand at the origin.

▶ **Definition 38.**

$$\pi(x) = \sum\nolimits_{p\leq x} 1 = |\{p \mid p \text{ prime} \wedge p \leq x\}| \qquad p_n = \text{the } n\text{-th prime number } (p_0 = 2)$$

$$\vartheta(x) = \sum\nolimits_{p\leq x} \ln p \qquad \mathfrak{M}(x) = \sum\nolimits_{p\leq x} \ln p/p$$

$$\psi(x) = \sum\nolimits_{n\leq x} \Lambda(n) = \sum\nolimits_{p^k\leq x} \ln p \qquad M(x) = \sum\nolimits_{n\leq x} \mu(n)$$

$\pi(x)$ is usually called the "prime-counting function". $\vartheta(x)$ and $\psi(x)$ are the first and the second Chebyshev function. $\mu(n)$ is the Möbius μ function. $\mathfrak{M}(x)$ is a non-standard notation I adopted; the function that it denotes is related to Mertens' first theorem and a key part in Newman's proof of the PNT.

▶ **Theorem 39.** *The following are all equivalent formulations of the PNT, i. e. given one of them, it is fairly easy to show the other ones by elementary means:*

$$\pi(x) \sim x/\ln x \qquad \pi(x)\ln\pi(x) \sim x \qquad p_n \sim n\ln n \qquad \vartheta(x) \sim x \qquad \psi(x) \sim x \qquad M(x) \in o(x)$$

Most of these equivalence proofs are quite short, both on paper and in Isabelle.

Newman's approach to prove the PNT is then to prove $\mathfrak{M}(x) = \ln x + c + o(1)$, which implies $\vartheta(x) \sim x$ fairly directly as we shall see. The key ingredient is a Tauberian theorem first proven by Ingham, which we will discuss now.

6.1 Ingham's Tauberian Theorem

A Tauberian theorem is a theorem that allows one to show – under certain conditions – that a series converges in some region if the function that it defines exists there. In our case, Ingham's theorem allows us to show that certain Dirichlet series converge not just to the right of the abscissa of convergence, but *on it* as well. The precise statement is as follows:

▶ **Theorem 40** (Ingham's Tauberian theorem)**.** *Let $F(s) = \sum a_n n^{-s}$ be a Dirichlet series with $a_n \in O(n^{\sigma-1})$ for some $\sigma \in \mathbb{R}$. Then F converges to an analytic function $f(s)$ for* $\mathrm{Re}(s) > \sigma$*. If $f(s)$ is analytic on the larger set* $\mathrm{Re}(s) \geq \sigma$*, then F also converges to $f(s)$ for all* $\mathrm{Re}(s) \geq \sigma$*.*

One can w. l. o. g. assume $\sigma = 1$. Newman then proves the theorem by applying the Residue Theorem twice, once to a circle around 0 with a vertical cut-off line to the left of the origin, close to the abscissa of convergence (see Figure 2) and once to a full circle around the origin.

My formal proof follows Newman's argument very closely, but like Harrison, I use a modified version of Newman's contour: a semicircle plus a rectangle (see Figure 2). The value of the integral is the same in both cases since the two contours are homotopic, but the bounding of the contributions of the various parts of the contour is different.

The reason why I picked Harrison's contour over Newman's is that I could not understand how Newman's bounding of the different contributions fits to his contour, and it seems likely that this is also the reason why Harrison altered the contour in the first place. Additionally, the shape of the inside of Harrison's contour is somewhat easier to describe.

The formal proof is quite short (roughly 500 lines) and was – apart from the issue I just mentioned – very straightforward to write. However, it again suffers from the aforementioned typical problems of complex analysis in Isabelle, namely having to prove many side conditions such as the geometry of the integration contours. The winding numbers, on the other hand, are unproblematic this time since the contours are very simple.

6.2 An Overview of the Remainder of Newman's Proof

Recall that our main objective was to prove

$$\mathfrak{M}(x) \sim \ln x + c + o(1)\ . \tag{5}$$

The starting point is Mertens' First Theorem, which I prove following e. g. Hildebrand [21]:

▶ **Theorem 41** (Mertens' First Theorem). $\mathfrak{M}(x) = \ln x + O(1)$

To then show (5) from this, Newman defines the Dirichlet series $f(s) := \sum_{n=1}^{\infty} \mathfrak{M}(n) n^{-s}$. Since $\mathfrak{M}(n) - \ln n$ is bounded, $f(s)$ converges absolutely for $\mathrm{Re}(s) > 1$. Rearrangement yields

$$f(s) = \sum_p \frac{\ln p}{p} \zeta(s,p) \qquad \text{for } \mathrm{Re}(s) > 1$$

and further rearrangements show

$$f(s) = \frac{A(s) - \zeta'(s)/\zeta(s)}{s-1} \qquad \text{for } \mathrm{Re}(s) > 1$$

for some function $A(s)$ that is analytic for $\mathrm{Re}(s) > \frac{1}{2}$. Moreover, $\zeta'(s)/\zeta(s)$ is analytic for $\mathrm{Re}(s) \geq 1$, $s \neq 1$ due to the non-vanishing of $\zeta(s)$ in that domain (cf. Theorem 30).

Putting everything together, we obtain that $f(s)$ can be continued analytically to $\mathrm{Re}(s) \geq 1$ except for a double pole at $s = 1$. As Newman states, this double pole can be turned into a simple pole by adding $\zeta'(s)$, and that simple pole can then be eliminated by subtracting a suitable multiple of $\zeta(s)$, yielding a function $g(s) := f(s) + \zeta'(s) - c\,\zeta(s)$ that is analytic for $\mathrm{Re}(s) \geq 1$ and has the Dirichlet series

$$g(s) = \sum_{n=1}^{\infty} \underbrace{(\mathfrak{M}(n) - \ln n - c)}_{=:\,a_n} n^{-s}\ .$$

Applying Theorem 40, we deduce that this series converges for $\mathrm{Re}(s) \geq 1$. For $s = 1$, this means that $\sum_{n=1}^{\infty} \frac{a_n}{n}$ is summable. Next, Newman proves the following lemma:

▶ **Lemma 42.** *Let $a_n : \mathbb{N} \to \mathbb{R}$ be non-decreasing and $\sum_{n=0}^{\infty} \frac{a_n}{n}$ be summable. Then $a_n \longrightarrow 0$.*

Applied to our a_n from before, we get $\mathfrak{M}(n) - \ln n \longrightarrow c$. From this, the slightly stronger version on real numbers (5) follows easily by noting that $\ln x - \ln\lfloor x \rfloor \longrightarrow 0$.

There were no major difficulties in formalising any of this. However, some parts deserve a few comments:

- The rearrangements leading to the analytic continuation of $f(s)$ involve changing the order of summation in nested infinite sums. To do this, I used Isabelle's library for absolutely summable families. This makes the arguments nice to formalise, but the library has the problem of having a function for the *value* of an infinite sum and for its *existence*. Any rearrangement of sums therefore has to be done twice, once for the value of the sum and once for its summability. Similar problems occur in Isabelle with nested integrals and it is not clear if and how this can be avoided in a HOL-based theorem prover.
- Showing that $A(s)$ is indeed analytic for $\mathrm{Re}(s) > \frac{1}{2}$ was a surprisingly easy application of the *Weierstraß M test* with the bounding series $M_n := \ln n (Cn^{-x-1} + n^{-x}(n^x - 1)^{-1})$. The proof obligation that M_n be summable can be solved by showing $M_n \in O(n^{-1-\varepsilon})$ with a suitable $\varepsilon > 0$, and this can be shown by Isabelle's automation for real limits [17].
- The pole cancellation argument showing that $g(s)$ is analytic is about 86 lines long, which is not too long, but still longer than one might expect given that it is obvious considering the Laurent series expansions of the functions involved. This is due to the fact that there is currently no theory of Laurent series expansions in Isabelle yet. In the future, this entire argument could potentially be automated by computing Laurent series expansions for meromorphic functions similarly to how Isabelle's automation already computes Multiseries expansions [17] for real-valued functions.
- The proof of Lemma 42 is very technical and tedious, but it seems to me that this is the case in Newman's paper presentation as well.

The last remaining step, showing that $\mathfrak{M}(x) - \ln x \longrightarrow c$ implies $\vartheta(x) \sim x$, is left as an exercise to the reader by Newman. Harrison was not quite sure what Newman meant [20] and proceeded to prove a number of very technical and ad-hoc lemmas that I find very difficult to follow. Therefore, instead of attempting to port Harrison's proof, I followed Newman's hint in the book and used Abel's summation formula to write $\vartheta(x)$ in terms of $\mathfrak{M}(x)$:

$$\vartheta(x) = x\mathfrak{M}(x) - \textstyle\int_2^x \mathfrak{M}(t)\,\mathrm{d}t \tag{6}$$

Substituting (5) into (6) yields, in a straightforward way,

$$\begin{aligned}\vartheta(x) &= x\ln x + cx + o(x) - \textstyle\int_2^x \ln t + c + o(1)\,\mathrm{d}t\\ &= x\ln x + cx + o(x) - (x\ln x - x + cx + o(x)) = x + o(x)\end{aligned}$$

and thus the desired $\vartheta(x) \sim x$. I find it likely that this is what Newman had in mind. ◀

7 Various Other Interesting Results

In this last section, I will give a few examples of other interesting number-theoretic results that I have formalised. The proofs were all fairly straightforward and there is not much to be said about them, but they are worth mentioning nonetheless.

▶ **Theorem 43** (Dirichlet's Theorem). *Let $m > 0$ and $\gcd(k, m) = 1$. Then there are infinitely many primes congruent k modulo m.*

▶ **Theorem 44** (Elementary bounds for $\pi(x)$ and p_n). *For any $x \geq 2$ and $n > 0$, we have:*

$$\frac{1}{6}\frac{x}{\ln x} < \pi(x) < 3(e^{-1} + \ln 2)\frac{x}{\ln x} \qquad \textit{and} \qquad \frac{139}{443} n \ln n \leq p_{n-1} < 12(n \ln n + n \ln(12/e))$$

In particular, this implies $\pi(x) \in \Theta(x/\ln x)$ and $p_n \in \Theta(n \ln n)$. All of this can be derived without the PNT (hence "elementary" results).

▶ **Theorem 45** (Mertens' three theorems).

- $-1 - 9\pi^{-2} < \mathfrak{M}(n) - \ln n \leq \ln 4$ *for all $n > 0$ and thus $|\mathfrak{M}(n) - \ln n| < 2$.*
- $|(\sum_{p \leq x} 1/p) - \ln\ln x - M| \leq 4/\ln x$ *for all $x \geq 2$ and thus* $\sum_{p \leq x} 1/p = \ln\ln x + M + O(1/\ln x)$ *where M is the Meissel–Mertens constant.*
- $\prod_{p \leq x}(1 - 1/p) = C/\ln x + O(\ln^{-2} x)$ *for some constant $C > 0$.*

Typically, number-theoretic functions that talk about a *single* integer such as $\varphi(n)$ and $\sigma_0(n)$ oscillate heavily and therefore have no nice asymptotics like $\pi(x) \sim x \ln x$. However, their *averages* (i. e. $\sum_{n \leq x} \varphi(n)$) are often more well-behaved:

▶ **Theorem 46** (Averages of arithmetical functions).

- *Let $S(x)$ denote the number of square-free integers $\leq x$. Then $S(x) = \frac{6}{\pi^2}x + O(\sqrt{x})$, i. e. $6/\pi^2 \approx 60.8\,\%$ of integers are square-free.*
- *Euler's totient function φ fulfils $\sum_{n \leq x} \varphi(n) = \frac{3}{\pi^2}x^2 + O(x \ln x)$, i. e. on average, an integer n has $\frac{3}{\pi^2}n$ numbers $\leq n$ that are coprime to it ($\approx 30.4\,\%$).*
- *The divisor function σ_0 fulfils $\sum_{n \leq x} \sigma_0(n) = x \ln x + (2\gamma - 1)x + O(\sqrt{x})$ where $\gamma \approx 0.5772$ is the Euler–Mascheroni constant, i. e. on average, an integer n has $\ln n + 2\gamma - 1$ divisors.*
- $\sum_{n \leq x} \sigma_\alpha(n) = \frac{\zeta(\alpha+1)}{\alpha+1}x^{\alpha+1} + O(R(x))$ *for $\alpha > 0$ where $R(x) = x \ln x$ if $\alpha = 1$ and $R(x) = x^{\max(1,\alpha)}$ otherwise.*
- $\sum_{n \leq x} \sigma_{-\alpha}(n) = \zeta(\alpha + 1)\, x + O(R(x))$ *for $\alpha > 0$ where $R(x) = \ln x$ if $\alpha = 1$ and $R(x) = x^{\max(0,1-\alpha)}$ otherwise.*

Lastly, the following are interesting consequences that follow relatively easily from the PNT:

▶ **Corollary 47.**

- *For each $c > 1$, there exists an x_0 s. t. all intervals $(x, cx]$ with $x \geq x_0$ contain a prime.*
- *The fractions of the form p/q for prime p, q are dense in $\mathbb{R}_{>0}$.*
- $\operatorname{lcm}(1, \ldots, n) = \exp(x + o(x))$
- $\limsup_{n\to\infty} \omega(n) \ln\ln n/\ln n = 1$
- $\limsup_{n\to\infty} \ln \sigma_0(n) \ln\ln n/\ln n = \ln 2$
- $\liminf_{n\to\infty} \varphi(n) \ln\ln n/n = C$ *for some $C \in \mathbb{R}_{>0}$*

The last three statements perhaps deserve some more explanation: They give asymptotic bounds for $\omega(n)$, $\sigma_0(n)$, and $\varphi(n)$. For instance, $\omega(n) < c \ln n/\ln\ln n$ for all sufficiently large n if $c > 1$, but $\omega(n) > c \ln n/\ln\ln n$ for infinitely many n if $c < 1$. Thus, $\ln n/\ln\ln n$ is the best possible upper bound of that shape for $\omega(n)$ (and analogously for the other two).

As for the other direction, recall that $\omega(p) = 1$, $\sigma_0(p) = 2$, and $\varphi(p) = p-1$. Therefore, the above results show that $\omega(n)$ oscillates between 1 and $\ln n/\ln\ln n$, $\sigma_0(n)$ oscillates between 2 and $2^{\ln n/\ln\ln n}$, and $\varphi(n)$ oscillates between $Cn/\ln\ln n$ and $n - 1$.

8 Size of the Formalisation

The formalised material is spread over five AFP entries [13, 12, 15, 18, 16]. They have a combined size of roughly 25,000 lines of Isabelle code, with the two largest single files by far being those on the analytic properties of Dirichlet series and the properties of the ζ functions.

With the exception of a few minor results, the work presented here was done in 1.5 years by one person – however, the work was not done continuously, but sporadically whenever I found time for it. The total amount of time that went into it is therefore difficult to measure. As a point of reference, the formalisation of Newman's proof of the Prime Number Theorem (with all the components such as Dirichlet series and the ζ function already in place) comprises 3300 lines and took 6 days of full-time work. However, I used two small lemmas that had previously been ported from Harrison's HOL Light formalisation by Paulson. Considering this, a time frame of 7 days for proving the Prime Number Theorem seems reasonable. Based on this, a total effort of 4–6 person-months for the entire work seems realistic.

The formalisation proceeded smoothly and without major difficulties, although some aspects of it stand out as considerably more painful than one might hope:

1. applying the Residue Theorem
2. geometric properties of integration contours
3. manipulating nested infinite sums
4. establishing homotopy of concrete composite paths
5. reasoning about cancellation of poles

For the first three items, it is not clear to me if and how this can be improved – or if, perhaps, there is simply an inherent difficulty in doing such things formally.

Item 4 could probably be addressed by providing more library results about homotopy.

Item 5 could be easily managed by building a tactic to automatically compute Laurent series expansions for meromorphic functions, similar to the existing one for Multiseries expansions of real functions [17]. This would be an interesting project for the future. Extending the limit automation to use not just full asympotic expansions but also partial asymptotic information (such as $\vartheta(x) \sim x$) would also occasionally eliminate some tedious manual work.

A related issue is that reasoning with asymptotic expansions like $f(x) = x^2 + \ln x + O(1/x)$ can be tedious in Isabelle/HOL. They *can* be written as $f =_o (\lambda x.\ x\,\hat{}\,2 + \ln x) +_o O(\lambda x.\ 1/x)$, but there is currently little support for working with them. Affeldt *et al.* [1] demonstrated an approach for this in Coq that could possibly be adapted to Isabelle/HOL.

9 Conclusion

I formalised a large portion of a mathematical textbook on an advanced topic, namely Analytic Number Theory. While some results from this field have been formalised before (such as Dirichlet's Theorem and the Prime Number Theorem), they typically tried to obtain a short route to the result without building an actual library of Analytic Number Theory.

In my opinion, this work demonstrates the following:

- Formalising an entire mathematical textbook in a modern theorem prover *can* be feasible with a moderate amount of effort.
- Good and extensive libraries (e. g. on complex analysis and Dirichlet series) can yield short, clear, and high-level proofs of "high-profile" results like the Prime Number Theorem.
- Specialised tools (e. g. for proving limits or computing winding numbers) are invaluable, as they can take care of tedious and uninteresting parts of the proofs and "close the gap" between what is obvious to a human mathematician and what is easy to do in the system.

There is already work in progress on formalising the remaining parts of Apostol's book. After that, a natural continuation would be to focus on the second volume of Apostol's book, which is called *Modular Functions and Dirichlet Series in Number Theory* [3]. This would be another big step in formalising the essential tools of modern number theory in a theorem prover.

References

1 Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization Techniques for Asymptotic Reasoning in Classical Analysis. *J. Formalized Reasoning*, 11(1):43–76, 2018. `doi:10.6092/issn.1972-5787/8124`.

2 Tom M. Apostol. *Introduction to analytic number theory*. Undergraduate Texts in Mathematics. Springer-Verlag, 1976. `doi:10.1007/978-1-4757-5579-4`.

3 Tom M. Apostol. *Modular Functions and Dirichlet Series in Number Theory*, volume 41 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1990. `doi:10.1007/978-1-4612-0999-7`.

4 Andrea Asperti and Wilmer Ricciotti. A proof of Bertrand's postulate. *Journal of Formalized Reasoning*, 5(1):37–57, 2012. `doi:10.6092/issn.1972-5787/3406`.

5 Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A Formally Verified Proof of the Prime Number Theorem. *ACM Trans. Comput. Logic*, 9(1), December 2007. `doi:10.1145/1297658.1297660`.

6 Raymond Ayoub. Euler and the zeta function. *The American Mathematical Monthly*, 81(10):1067–1086, 1974. `doi:10.2307/2319041`.

7 Joseph Bak and Donald J. Newman. *Complex Analysis*. Undergraduate Texts in Mathematics. Springer New York, 1999.

8 Clemens Ballarin. Locales: A Module System for Mathematical Theories. *Journal of Automated Reasoning*, 52(2):123–153, 2014. `doi:10.1007/s10817-013-9284-7`.

9 Julian Biendarra and Manuel Eberl. Bertrand's postulate. *Archive of Formal Proofs*, January 2017. , Formal proof development. URL: `http://isa-afp.org/entries/Bertrands_Postulate.html`.

10 Mario Carneiro. Arithmetic in Metamath, Case Study: Bertrand's Postulate. *CoRR*, abs/1503.02349, 2015. `arXiv:1503.02349`.

11 Mario Carneiro. Formalization of the prime number theorem and Dirichlet's theorem. In *Proceedings of the 9th Conference on Intelligent Computer Mathematics (CICM 2016)*, pages 10–13, 2016. URL: `http://ceur-ws.org/Vol-1785/F3.pdf`.

12 Manuel Eberl. Dirichlet L-functions and Dirichlet's theorem. *Archive of Formal Proofs*, December 2017. , Formal proof development. URL: `http://isa-afp.org/entries/Dirichlet_L.html`.

13 Manuel Eberl. Dirichlet series. *Archive of Formal Proofs*, October 2017. , Formal proof development. URL: `http://isa-afp.org/entries/Dirichlet_Series.html`.

14 Manuel Eberl. The Euler–MacLaurin Formula. *Archive of Formal Proofs*, March 2017. , Formal proof development. URL: `http://isa-afp.org/entries/Euler_MacLaurin.html`.

15 Manuel Eberl. The Hurwitz and Riemann ζ Functions. *Archive of Formal Proofs*, October 2017. , Formal proof development. URL: `http://isa-afp.org/entries/Zeta_Function.html`.

16 Manuel Eberl. Elementary Facts About the Distribution of Primes. *Archive of Formal Proofs*, February 2019. , Formal proof development. URL: `http://isa-afp.org/entries/Prime_Distribution_Elementary.html`.

17 Manuel Eberl. Verified Real Asymptotics in Isabelle/HOL. Draft available at `https://www21.in.tum.de/~eberlm/real_asymp.pdf`, 2019.

18 Manuel Eberl and Lawrence C. Paulson. The Prime Number Theorem. *Archive of Formal Proofs*, September 2018. , Formal proof development. URL: `http://isa-afp.org/entries/Prime_Number_Theorem.html`.

19 John Harrison. A formalized proof of Dirichlet's theorem on primes in arithmetic progression. *Journal of Formalized Reasoning*, 2(1):63–83, 2009. `doi:10.6092/issn.1972-5787/1558`.

20 John Harrison. Formalizing an analytic proof of the Prime Number Theorem (Dedicated to Mike Gordon on the occasion of his 60th birthday). *Journal of Automated Reasoning*, 43(3):243–261, October 2009. `doi:10.1007/s10817-009-9145-6`.

21 A. J. Hildebrand. Introduction to analytic number theory (lecture notes). `https://faculty.math.illinois.edu/~hildebr/ant/`.

22 Johannes Hölzl, Fabian Immler, and Brian Huffman. Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 279–294. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-39634-2_21.

23 Wenda Li and Lawrence C. Paulson. Evaluating Winding Numbers and Counting Complex Roots through Cauchy Indices in Isabelle/HOL. *CoRR*, abs/1804.03922, 2018. arXiv:1804.03922.

24 M. Ram Murty and Marilyn Reece. A simple derivation of $\zeta(1-K) = -B_K/K$. *Funct. Approx. Comment. Math.*, 28:141–154, 2000. doi:10.7169/facm/1538186691.

25 Donald J. Newman. *Analytic number theory.* Number 177 in Graduate Texts in Mathematics. Springer, 1998. doi:10.1007/b98872.

26 Marco Riccardi. Pocklington's theorem and Bertrand's postulate. *Formalized Mathematics*, 14:47–52, January 2006. doi:10.2478/v10037-006-0007-y.

27 Laurent Théry. Proving Pearl: Knuth's Algorithm for Prime Numbers. In David Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics*, pages 304–318, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/10930755_20.

D Linear Recurrences

This chapter was originally published as an article in the proceedings of a peer-reviewed conference:

> M. Eberl.
> Verified Solving and Asymptotics of Linear Recurrences.
> In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs.*
> CPP 2019, pages 27–37, New York, NY, USA, 2019. ACM.
> DOI: `10.1145/3293880.3294090`.

I am the sole author of this article, thus all contributions are mine.

Synopsis: This work provides a formalisation in Isabelle/HOL of the theory of linear recurrences with constant coefficients and rational generating functions, and the connection between these two. It also provides a fully executable solver to determine the closed-form solution of a recurrence and a fast checker to certify 'Big-O' bounds for a solution without computing it.

On the following pages, the full article is reproduced in its published form in accordance to the ACM author rights for reproduction in a dissertation. The official version in the ACM Digital Library can be found under the DOI cited above.

Verified Solving and Asymptotics of Linear Recurrences

Manuel Eberl
Institut für Informatik
Technische Universität München
Garching bei München, Germany
eberlm@in.tum.de

Abstract

Linear recurrences with constant coefficients are an interesting class of recurrence equations that can be solved explicitly. The most famous example are certainly the Fibonacci numbers with the equation $f(n) = f(n-1) + f(n-2)$ and the quite non-obvious closed form

$$\frac{1}{\sqrt{5}}(\varphi^n - (-\varphi)^{-n})$$

where φ is the golden ratio.

This work builds on existing tools in Isabelle – such as formal power series and polynomial factorisation algorithms – to develop a theory of these recurrences and derive a fully executable solver for them that can be exported to programming languages like Haskell.

Based on this development, I also provide an efficient method to prove 'Big-O' asymptotics of a solution automatically without explicitly finding the closed-form solution first.

CCS Concepts • **Mathematics of computing → Generating functions**; *Solvers*;

Keywords linear recurrences, generating functions, asymptotics, formal power series, Fibonacci, Isabelle, theorem proving

ACM Reference Format:
Manuel Eberl. 2019. Verified Solving and Asymptotics of Linear Recurrences. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19), January 14–15, 2019, Cascais, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3293880.3294090

1 Introduction

This paper is about verifying the theory, the asymptotics, and an executable solver for linear recurrences with constant coefficients. It supports both homogeneous recurrences and inhomogeneous recurrences where the inhomogeneous part is from a certain class. From this point onward, I will use the term 'linear recurrence' and implicitly mean 'linear recurrence in one variable with constant coefficients'.

The most famous such recurrence is certainly the one defining the *Fibonacci numbers*:

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

They are named after the 12th-century Italian mathematician Leonardo of Pisa – who is nowadays known better by the name *Fibonacci* – although it was studied by Indian mathematicians much earlier in connection with the possible patterns in the metre of Sanskrit prosody. Fibonacci, on the other hand, presented them in the context of a puzzle about the population growth of rabbits: assuming a pair of adult rabbits produces a new pair of baby rabbits every month, rabbits take one month to mature, and rabbits never die, what is the number of adult rabbit pairs after n months, starting with a single pair of baby rabbits?

The Fibonacci numbers have a number of very interesting properties and occur in many places in mathematics. They can be written as the closed-form expression

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

where $\varphi = \frac{1}{2}(1+\sqrt{5})$ is the golden ratio and $\psi = 1 - \varphi = \frac{1}{2}(1-\sqrt{5})$.

This may seem surprising since the closed form contains irrational constants that do not cancel in an obvious way even though the Fibonacci numbers are all natural numbers.

This closed form directly implies the asymptotic estimate $f(n) \sim \varphi^n/\sqrt{5}$, which is very accurate – it turns out, in fact, that $f(n) = [\varphi^n/\sqrt{5}]$ for all positive n, where $[\cdot]$ is the 'nearest integer' function.

More generally, such recurrences arise in certain enumeration problems: the number of steps required to solve the 'Tower of Hanoi', the number of ordered partitions of integers, or enumerating all lists of a given length with some additional restrictions (e. g. forbidden patterns). They also arise in the average-time analysis of recursive algorithms [6], in the analysis of imperative programs with loops [12, 17], and in the analysis of probabilistic algorithms like random walks[11].

It is therefore of great interest that these recurrences can be solved automatically. The key is the so-called *characteristic polynomial*, which can be read off directly from the recurrence equation. The roots of this polynomial determine the general shape of the solution, whereas the precise coefficients depend on the initial values. For example, the characteristic polynomial of the Fibonacci recurrence $F_n = F_{n-1} + F_{n-2}$ is $x^2 - x - 1$, whose roots are φ and ψ, so that the solution must have the general form $c_1\varphi^n + c_2\psi^n$ independently of the initial values. The initial values do, however, determine the values of c_1 and c_2.

Developing the theory behind this in Isabelle/HOL can be done in the most nice and abstract way using *formal power series* (FPSs); in particular, using the well-known correspondence between linear recurrences with constant coefficients and rational FPSs, i. e. power series that are of the form p/q for complex polynomials p, q with $q \neq 0$.

In the end, I will use some existing Isabelle tools to derive a fully automatic solver for linear recurrences with constant coefficients. Furthermore, I provide a more efficient tool to certify 'Big-O' asymptotic bounds for the solution of such a recurrence. To my knowledge, this is the first general treatment of linear recurrences in a proof assistant. The development is available as an entry in the *Archive of Formal Proofs* [4]. At the time of writing, a part of it is only available in the *development version* of the Archive (https://devel.isa-afp.org).

2 Mathematical Basics

2.1 Formal Power Series

A key ingredient in the textbook approach to solving linear recurrences are *formal power series* (FPSs) in the form of *ordinary generating functions*. These are purely formal objects; their purpose is nothing but to have a nice algebraic object that represents a sequence.

Formally, the commutative (semi-)ring of FPSs over a commutative (semi-)ring R with formal parameter X is written as $R[[X]]$, and its elements are written e. g. as

$$A(X) = \sum_{n=0}^{\infty} a_n X^n ,$$

where a_n is the sequence of coefficients. Let us denote a_n – the n-th coefficient of the power series A – with $[X^N]\,A$, and call $a_0 = [X^0]\,A$ the *constant coefficient*. In analogy to polynomials, the basic operations are defined as:

$$\begin{aligned} 0 &= \sum_{n=0}^{\infty} 0 \cdot X^n \\ 1 &= \sum_{n=0}^{\infty} (\text{if } n = 0 \text{ then } 1 \text{ else } 0) X^n \\ A(z) + B(z) &= \sum_{n=0}^{\infty} (a_n + b_n) X^n \\ A(z) \cdot B(z) &= \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n} a_i b_{n-i} \right) X^n \end{aligned}$$

From now one, we shall always restrict ourselves to the case where the underlying ring is a field K, and at some point, we will also introduce the additional assumption that K is algebraically closed. In practice, this field will therefore be the complex numbers, but the theory is developed as generally as possible.

In the polynomial ring $K[X]$, the only units (i. e. elements with a multiplicative inverse) are then the constant non-zero polynomials. In $K[[X]]$, on the other hand, all FPSs with non-zero constant coefficient are invertible with

$$A(z)^{-1} = \sum_{n=0}^{\infty} b_n X^n \quad \text{for } b_n = \begin{cases} \frac{1}{a_0} & \text{for } n = 0 \\ -\frac{1}{a_0} \sum_{i=1}^{n} a_i b_{n-i} & \text{otherwise} \end{cases}$$

It is clear that there exists an injective canonical homomorphism $K[X] \longrightarrow K[[X]]$. One therefore implicitly identifies polynomials in $K[X]$ with the corresponding FPS in $K[[X]]$. Furthermore, if $p, q \in K[X]$ and $q(0) \neq 0$ (i. e. the constant coefficient of q is non-zero), then q is a unit in $K[[X]]$ and the quotient $p(x)/q(x) \in K[[X]]$ is well-defined. Let us call FPSs of this form *rational*, and they form a sub-ring of $K[[X]]$. Algebraically speaking, this ring of rational FPSs is the localisation of $K[X]$ w. r. t. $\{p \mid p(0) \neq 0\}$.

Note that this ring of rational FPSs is smaller than the ring of rational functions $K(X)$: The latter contains e. g. $1/X$, whereas the former does not. The rational FPSs are essentially those elements of $K(X)$ that do not have a pole at the origin.

This sub-ring is what we are interested in, since rational FPSs are precisely those whose coefficients satisfy linear recurrences. I therefore defined the Isabelle type *ratfps* that corresponds to this sub-ring and connected it to the FPS type with the injective canonical homomorphism α *ratfps* $\longrightarrow$ α *fps* (where α is the type of the coefficients) on one side and to the fraction field of polynomials with an injective homomorphism α *ratfps* $\longrightarrow$ α *poly fract* on the other side. Isabelle's Code Generator was set up to perform computations on rational FPSs by representing the quotient as a pair of coprime polynomials where the second one (the denominator) must be monic and non-zero (which is a convenient unique representation). This makes it possible to write the solving algorithms in a very abstract way, operating directly on FPSs, and still directly generate executable code from these definitions.

A standard approach to solve recurrences is the following:

1. Use the recurrence to find a simple equation that characterises the FPS of the sequence.
2. Solve that equation to find a closed-form expression for the FPS.
3. Use algebraic transformations to break down the FPS into simpler parts.
4. Read off the solution as the coefficients of these simple parts.

In our case, more concretely, the steps are the following:

1. Write the inhomogeneous part as an FPS (in our case: as a rational FPS).
2. Use this to write the entire recurrence as a (rational) FPS.
3. Factor the denominator of the FPS into linear polynomials of the form $1 - c_i X$.
4. Perform *Partial Fraction Decomposition* to bring this rational FPS into the form.
$$p(X) + \sum_{i=1}^{k} \frac{b_i}{(1 - c_i X)^{k_i}}$$
for a polynomial $p \in K[X]$ and numbers $b_i, c_i \in K$.
5. Read off the coefficients from this sum.

2.2 The Homogeneous Part

Let us first consider a homogeneous recurrence. To obtain a more uniform presentation, let us assume that it is given in the form

$$\forall n \geq l.\ c_0 f(n) + c_1 f(n+1) + \ldots + c_m f(n+m) = 0 \qquad (1)$$

with $m + l$ base cases (otherwise, the sequence would not be uniquely defined). The number l is the number of 'excess' base cases and will usually be 0.

Let $F(X)$ be the generating function of f and define the polynomial q as

$$q(X) = c_0 X^m + c_1 X^{m-1} + \ldots + c_m \ .$$

Then, for any $n \geq m + l$, we have:

$$\begin{aligned}
&[X^n]\,(q(X)F(X)) \\
&\quad = [X^n]\,\big(c_0 X^m F(X) + c_1 X^{m-1} F(X) + \ldots + c_m F(X)\big) \\
&\quad = c_0 f(n-m) + c_1 f(n-m+1) + \ldots + c_m f(n) \overset{(1)}{=} 0
\end{aligned}$$

Therefore, $p(X) := q(X)F(X)$ is a polynomial of degree at most $m + l$. Its coefficients can easily be calculated as

$$[X^n]p(X) = \sum_{i=0}^{\min(m,n)} c_{m-i} f(n-i) \ .$$

In other words, we now have $F(X) = p(X)/q(X)$. Note that if $c_m \neq 0$, which is a reasonable demand to make from a well-formed recurrence equation, the constant coefficient of the denominator $q(X)$ is also non-zero.

2.3 The Inhomogeneous Part

Treating inhomogeneous recurrences can be done in a very similar way as homogeneous ones. In analogy to (1), we now consider recurrences of the form

$$\forall n \geq l.\ c_0 f(n) + c_1 f(n+1) + \ldots + c_m f(n+m) = g(n+m) \quad (2)$$

We again let $q(X)$ be as before and obtain:

$$\begin{aligned}
&[X^n]\,(q(X)F(X)) \\
&\quad = [X^n]\,\big(c_0 X^m F(X) + c_1 X^{m-1} F(X) + \ldots + c_m F(X)\big) \\
&\quad = c_0 f(n-m) + c_1 f(n-m+1) + \ldots + c_m f(n) \overset{(2)}{=} g(n)
\end{aligned}$$

If we let $G(X)$ be the generating function of the inhomogeneous part g, we therefore know that $p(X) := q(X)F(X) - G(X)$ is a polynomial of degree at most $m + l - 1$, and, analogously to before, its coefficients can be calculated as

$$[X^n]p(X) = \left(\sum_{i=0}^{\min(m,n)} c_{m-i} f(n-i)\right) - g(n)$$

and we have $F(X) = (p(X) + G(X))/q(X)$. In particular, this means that if $G(X)$ is rational, $F(X)$ is as well.

The remaining question is how to go from the closed form of the sequence $g(n)$ to a rational generating function $G(X)$. However, since $G(X)$ is rational iff $g(n)$ can be written as a sum of terms of the form $cn^k a^n$ for $c, a \in \mathbb{C}$ and $k \in \mathbb{N}$, it suffices to determine what the generating function of $n^k a^n$ is. For this purpose, let $B_k(X) := \sum_{n=0}^{\infty} n^k X^n$. We obviously have:

$$\sum_{n=0}^{\infty} n^k a^n X^n = B_k(aX)$$

We therefore only have to determine $B_k(X)$. As it turns out,

$$B_k(X) = \begin{cases} 1/(1-X) & \text{for } k = 0 \\ X E_k(X)/(1-X)^{k+1} & \text{otherwise} \end{cases}$$

where $E_k(X)$ is the k-th Eulerian polynomial, defined as:

$$E_k(X) = \begin{cases} 1 & \text{for } k = 0 \\ (nX - X + 1)\, E_{k-1}(X) & \text{otherwise} \\ \quad + (X - X^2)\, E'_{k-1}(X) & \end{cases}$$

This is easily verified by induction over k. In conclusion, we have

$$\sum_{n=0}^{\infty} n^k a^n X^n = \begin{cases} 1/(1-aX) & \text{for } k = 0 \\ aX E_k(aX)/(1-aX)^{k+1} & \text{otherwise} \end{cases}$$

and can therefore write the inhomogeneous part as a rational FPS as long as it is given in polynomial-exponential form.

From now on, we will consider the following recurrence as a running example:

$$f(n) - f(n-1) - 2f(n-2) = n\,2^n \qquad f(0) = f(1) = 0$$

By the above result, the generating function of the inhomogenous part $n\,2^n$ is $G(X) = 2X/(1-2X)^2$ (since $E_1 = 1$). According to the above definitions, we compute $p(X) = -2X$ and $q(X) = -2X^2 - X + 1$, leading to the generating function

$$\sum f(n)X^n = \frac{p(X)+G(X)}{q(X)} = \frac{8X^3-8X^2}{8X^4-4X^3-6X^2+5X-1}\ .$$

2.4 Factoring the Denominator

We now have the generating function for the recurrence in the form $p(X)/q(X)$ with $q(0) \neq 0$, and w. l. o. g. we can assume $q(X)$ to be monic. The next step is to factor $q(X)$ in order to break down the generating function into easier terms.

If the field K is algebraically closed, the $q(X)$ can always be factored into the form

$$d\,(1-c_1X)^{n_1}\dots(1-c_kX)^{n_k} \qquad \text{(for } d, c_i \in K\text{)}$$

Thiemann *et al.* [13, 14] have implemented several methods of factoring real and complex polynomials in Isabelle/HOL, using an implementation of algebraic real numbers based on *Sturm sequences.* Their algorithm produces a factorisation in terms of linear factors of the form $X - c$; applying it to q^R (the reflected polynomial) yields a factorisation in terms of linear factors $1 - cX$, as desired.

For our running example, factoring the denominator into such linear factors yields

$$8X^4 - 4X^3 - 6X^2 + 5X - 1 = -(1+X)(1-2X)^3\ .$$

2.5 Partial Fraction Decomposition

We now have our FPS in the form

$$\frac{p}{(1-c_1X)^{n_1}\dots(1-c_kX)^{n_k}}$$

and we want to break this up into simpler summands. This can be done with *Partial Fraction Decomposition*, which operates, more generally, on a quotient of the form

$$\frac{p}{q_1^{n_1}\dots q_k^{n_k}}$$

where all the q_k are pairwise coprime. Partial Fraction Decomposition then brings this quotient into the form

$$r + \sum_{i=1}^{k}\sum_{j=1}^{n_i}\frac{s_{ij}}{q_i^j}$$

where $r, s_{ij} \in K[X]$ and each s_{ij} has a degree less than that of q_i. In particular, if the q_i have degree 1 (which is the case here), the s_{ij} must all be constants.

To do this, consider the fraction $p/(qr)$ for $\gcd(q,r) = 1$. Then the extended Euclidean algorithm gives us $s, t \in K[X]$ with $sq + tr = 1$ and therefore

$$\frac{p}{qr} = \frac{pt}{q} + \frac{ps}{r}$$

Iterating this process on our original quotient gives us the following decomposition:

$$\frac{p}{q_1^{n_1}\dots q_k^{n_k}} = \sum_{i=1}^{k}\frac{r_i}{q_i^{n_i}} \tag{3}$$

Iterated polynomial division by q_i on each summand yields

$$\begin{aligned}\frac{r_i}{q_i^{n_i}} &= \frac{s_{i,n_i} + q_i s_{i,n_i-1} + \ldots + q_i^{n_i-1}s_{i,1} + q_i^{n_i}s_{i,0}}{q_i^{n_i}}\\ &= \frac{s_{i,n_i}}{q_i^{n_i}} + \frac{s_{i,n_i-1}}{q_i^{n_i-1}} + \ldots + \frac{s_{i,1}}{q_i} + s_{i,0}\end{aligned}$$

Collecting the $s_{i,0}$ into a single polynomial r then gives us the desired form.

Alternatively, one can avoid the polynomial division in the last step: Since the polynomials q_i have the form $1 - cX$ in our case, the summands in (3) have the form

$$\frac{r_{n-1}X^{n-1} + \ldots + r_0}{(1-cX)^n}$$

and we can apply a Binomial transform to the r_i to express the numerator as a polynomial in $1 - cX$. This may yield slightly better performance and could be implemented as future work, but the n_i are typically relatively low anyway.

Applied to our running example, the decomposition is:

$$-\frac{8X^3-8X^2}{(1+X)(1-2X)^3} = \frac{\frac{16}{27}}{1+X} + \frac{\frac{2}{3}}{(1-2X)^3} - \frac{\frac{4}{9}}{(1-2X)^2} - \frac{\frac{22}{27}}{1-2X}$$

2.6 Constructing the Solution

We now have the FPS in the form

$$r(X) + \sum_{i=1}^{k}\sum_{j=1}^{n_0}\frac{a_{ij}}{(1-c_iX)^j}\ .$$

To find the closed-form solution of the original recurrence, we must now extract the coefficients from this FPS. We can again do this for every summand individually.

For the polynomial summand r, the solution is obvious: The n-th coefficient of r as an FPS is simply the n-th coefficient of the polynomial r. If n is larger than the degree of r, we have $[X^n]\,r = 0$. The polynomial summand r can therefore be seen as an adjustment term that influences only the first few elements of the sequence. This is necessary when the recurrence is 'overspecified', i. e. there are more initial conditions given than necessary to define the sequence uniquely.

For the other summands, it suffices to consider the FPS $(1 - cX)^{-j}$ for $j > 0$. Using the FPS form of the generalised

Binomial Theorem, we find that

$$\begin{aligned}(1-cX)^{-j} &= \sum_{n=0}^{\infty} \binom{-j}{n}(-cX)^n \\ &= \sum_{n=0}^{\infty} c^n \binom{j+n-1}{n} X^n \\ &= \sum_{n=0}^{\infty} \frac{c^n}{(j-1)!}((n+1)\cdot \ldots \cdot (n+j-1))X^n\end{aligned}$$

Obviously, $p_j(n) := (n+1)\cdot \ldots \cdot (n+j-1)$ is a polynomial in n. In fact, it turns out that

$$p_j(n) = \sum_{i=0}^{j-1} s_{j,i+1} n^i$$

where $s_{i,n}$ are the Stirling numbers of the first kind, which gives a more efficient formula to compute the p_j. We can conclude:

$$[X^n]\,\frac{a_{ij}}{(1-c_iX)^j} = \frac{a_{ij}}{(j-1)!} p_j(n) c_i^n$$

With this, we now have a complete procedure starting from a homogeneous or inhomogeneous recurrence and ending with a concrete and computable representation of the closed-form solution of the recurrence.

Applying this to our running example, whose generating function we decomposed into

$$\frac{\frac{16}{27}}{1+X} + \frac{\frac{2}{3}}{(1-2X)^3} - \frac{\frac{4}{9}}{(1-2X)^2} - \frac{\frac{22}{27}}{1-2X}\,,$$

we obviously obtain the contribution $\frac{16}{27}\cdot(-1)^n$ for the first summand and $\frac{-22}{27}\cdot 2^n$ for the last one. For the other two summands, we compute

$$p_2(n) = n+1 \qquad p_3(n) = n^2+3n+2$$

and thereby:

$$[X^n]\,\frac{\frac{2}{3}}{(1-2X)^3} = \frac{\frac{2}{3}}{2!}(n^2+3n+2)2^n = \left(\frac{1}{3}n^2+n+\frac{2}{3}\right)2^n$$

$$[X^n]\,\frac{-\frac{4}{9}}{(1-2X)^2} = \frac{-\frac{4}{9}}{1!}(n+1)2^n = \left(-\frac{4}{9}n-\frac{4}{9}\right)2^n$$

Adding all the contributions together, we obtain the closed-form solution:

$$f(n) = \left(\frac{1}{3}n^2+\frac{5}{9}n-\frac{16}{27}\right)2^n + \frac{16}{27}\cdot(-1)^n$$

2.7 Asymptotics

Since the closed form of the coefficients of a rational FPS $p(X)/q(X)$ involves the complex roots of q, it can be somewhat unwieldy to work with: We have to fully factor the polynomial and do computations with (potentially complicated) algebraic numbers, which can lead to some problems, as we will see later. Fortunately, if we do not care about the precise closed form but only the asymptotics of the coefficients, there is an easier way.

Since any k-th order complex root z of the denominator $q(X)$ contributes a summand $r(n)z^{-n}$ with $\deg(r) \leq k-1$ to the solution, it is clear that the asymptotically dominant summands are those where $|z|$ is minimal. In fact, it is easy to see that whenever we can find a radius $R > 0$ such that there are no roots z with $|z| < R$ and all roots with $|z| = R$ have order $\leq k+1$, the solution is $O(n^k R^{-n})$. This fact has been proven in Isabelle.

Note that the numerator polynomial $p(X)$ does not appear in the asymptotics at all. In particular, this implies that the 'Big-O' bound for the solution of a linear recurrence holds irrespective of the precise initial values. However, the numerator polynomials (and thereby the initial values of a recurrence) *can* influence the asymptotics: If $p(X)$ and $q(X)$ have a non-trivial common divisor (i. e. they share at least one root), we can cancel that divisor from the fraction, which also reduces the number of roots in the denominator and may thereby lead to a smaller asymptotic upper bound.

When this is not possible – i. e. when the numerator and denominator are coprime, which can always be ensured for concrete polynomials in the numerator and denominator – then we have not only $\deg(r) \leq k$, but $\deg(r) = k$ (see e. g. Theorem IV.9 by Flajolet and Sedgewick [5]), which means that the above 'Big-O' estimate is tight. If there is a single dominant root, one therefore even gets a 'Big-Θ' bound. One could show this in Isabelle, but this was not done yet, since I considered the 'Big-O' bounds to be the most relevant ones for applications, e. g. in the analysis of algorithms.

For our running example, we can automatically prove that the solution is $O(n^2 2^n)$: The square-free factorisation of the denominator $8X^4-4X^3-6X^2+5X-1$ already yields the full factorisation $(1+X)(2X-1)^3$. We then consider the circle $|X| = \frac{1}{2}$ and note that none of the factors have a root inside that circle and no factor with exponent $> 2+1$ has a root on the circle (since no such factor exists). This certifies the upper bound $O(n^2 2^n)$.

3 Formalisation in Isabelle

3.1 Formal Power Series

I build on the existing formalisation of FPSs by Chaieb [3], which I extended, among other things, with a more general notion of division: Chaieb only defined division in the case where the divisor is a unit, i. e. has a non-zero constant coefficient. This is problematic because division can also be well-defined when the divisor is not a unit, e. g. X/X or $(X^3+X)/(X^2+X)$. I therefore defined the concept of a *subdegree* in Isabelle, i. e. the index of the first non-zero coefficient. The division of two FPSs is well-defined iff the subdegree of the divisor does not exceed that of the dividend, and the new division operator I defined works in all of these cases.

Before we proceed, the concept of *normalisation* in a ring in Isabelle must be explained. In mathematics, one usually defines the greatest common divisor of the integers 4 and 6 to be 2, even though −2 would also be a perfectly adequate choice. Similarly, for the polynomials $2X$ and $3X$ in the ring $\mathbb{R}[X]$, it makes sense to define the GCD to be just X, even though any cX for $c \in \mathbb{R} \setminus \{0\}$ would also be possible. The underlying problem is that concepts like GCD and LCM do not really operate on *elements* of a ring, but on *association classes*. However, it is, of course, usually more convenient to work with ring elements, which is why one designates a single element of an association class to be the canonical representative of that class.

In Isabelle, we capture this in the class *normalization_semidom*, which assumes the existence of a *normalize* operation that, given some element $x \in R$, returns the canonical representative of the association class of x. We call an element *normalized* if it is the representative of its association class.

For us, this is useful in handling fractions: I extended Chaieb's *fract* type, which implements the fraction field of a given integral domain R, by adding the concept of a *normalised fraction*. Rational numbers, for instance, can be brought into a unique normal form by requiring the numerator and denominator to be coprime and the denominator to be positive. The same thing can be done for any integral domain R with a GCD and a concept of normalisation: One can bring any fraction a/b into normal form by dividing a and b by their GCD and then normalising the resulting denominator and adjusting the numerator accordingly. This yields a fraction a'/b' such that a' and b' are coprime and b' is normalised, and this representation can easily be shown to be unique.

I then used this to define the type α *ratfps* as the subset of α *poly fract* on which the denominator of the normalised fraction has a non-zero constant coefficient. Using Isabelle's `code_abstype` feature and the *transfer* [8] package, operations on *ratfps* can be implemented in terms of pairs of polynomials with the above-mentioned invariant and thus make the *ratfps* type fully executable. This allows us to automatically translate our abstract algorithms on FPSs directly to executable code. As a bonus, it also makes the *ratfps* and *fps* types available to Isabelle's counterexample generator QuickCheck [2], which aids us by automatically providing counterexamples when we write down an incorrect theorem statement.

3.2 Partial Fraction Decomposition

I used the following very general view of Partial Fraction Decomposition: Let R be a Euclidean domain and S an arbitrary ring with a homomorphism $\varphi : R \longrightarrow S$. Let $n_1, \ldots, n_k \in \mathbb{N}_{>0}$ and $p, q_1, \ldots, q_k \in R$ such that the q_i are pairwise coprime and all the $\varphi(q_i)$ are units in S. Then, by following the process outlined in Section 2.5, we obtain $r, s_{ij} \in R$ such that

$$\frac{\varphi(p)}{\varphi(q_1)^{n_1} \ldots \varphi(q_k)^{n_k}} = \varphi(r) + \sum_{i=1}^{k} \sum_{j=1}^{n_i} \frac{\varphi(s_{ij})}{\varphi(q_i)^j}$$

and all the s_{ij} have a Euclidean norm less than the Euclidean norm of q_i.

In our case, the Euclidean domain R is the ring of polynomials $K[X]$, the ring S is the ring of rational FPSs, and φ is the canonical homomorphism *ratfps_of_poly*. Also, each of the q_i is of the form $1 - cX$, such that $\varphi(1 - cX)$ is always a unit in S.

However, thanks to this very general derivation, one could also use it for $R = \mathbb{Z}$ and $S = \mathbb{R}$ to bring $\frac{1}{90} = \frac{1}{2 \cdot 3^2 \cdot 5}$ into the form $-1 + \frac{1}{2} + \frac{1}{3^2} + \frac{2}{5}$.

3.3 Complex Root Counting

One step that was still left open in the discussion of coefficient asymptotics before was how to actually check the conditions that the polynomial has no roots inside a given circle around the origin, and all the roots on the circle itself have at most order k. To do this, we need two components:

- Wenda Li's executable root counting algorithm[9, 10] that can count the number of complex roots within certain subsets of the complex plane, e. g. circles, rectangles, and half-planes. Many variants are offered, but I use only root counting inside an open disc ($|z| < R$) and on a circle ($|z| = R$) without taking multiplicities into account.
- Square-free factorisation as formalised by Thiemann *et al.* [15, 16]. This is much less involved than a full factorisation and allows us to break up a polynomial into factors of the form $p_i(X)^{k_i}$ such that the p_i are square-free and pair-wise coprime. This means that each root of the original polynomial is present in exactly one of the p_i, and the corresponding k_i is the order of the root.

We therefore only have to run the square-free factorisation algorithm and then check that no p_i has a root with $|z| < R$, and that additionally no p_i with $k_i > k$ has any roots with $|z| = R$. None of this involves factoring the entire polynomial, and all computations can be done in the relatively pleasant field $\mathbb{Q}[i]$ – unless, of course, the coefficients or the radius R itself are already irrational themselves.

3.4 Code generation

The steps given above can be broken down into a number of Isabelle/HOL functions, which provides some modularity:

3.4.1 lhr_fps

This function takes a list of coefficients $c_0, \ldots, c_m$ and initial values $f_0, \ldots, f_{m+l-1}$ and returns the rational FPS of the corresponding recurrence as described in Section 2.2. Note that this implicitly uses the convention of Section 2.2 where the recurrence is only required to hold for $n \geq l$; i. e. extra initial values can be used to encode exceptions to the recurrence.

3.4.2 lir_fps

This is analogous to *lhr_fps*, but additionally takes a representation of the inhomogeneous part. This representation has the form of a list of tuples (a, b, k), encoding a sum over the terms $an^k b^n$. This is done according to the process outlined in Section 2.3.

3.4.3 solve_factored_ratfps

This function takes a rational FPS $F(X) = p(X)/q(X)$ where $q \neq 0$ and $q(X)$ has already been factored into terms of the form $1 - cX$ with $c \neq 0$ and returns a representation of a closed-form expression for its coefficients. This representation consists of a complex polynomial $a_{l-1}X^{l-1} + \ldots + a_0$ and a list of pairs $(p_i(X), b_i)$ where $p_i(X)$ is a complex polynomial and b_i is a complex number with the property that

$$[X^n]F(X) = a_n + \sum p_i(n) b_i^n$$

where $a_n = 0$ for all $n \geq l$. This is done by following the process described in Section 2.5 and immediately applying the process from Section 2.6 to the results.

3.4.4 solve_ratfps

The only missing link is now to factor the denominator polynomial of the rational FPS obtained from *lhr_fps* or *lir_fps* into terms of the form $1 - cX$. This is done by simply reflecting the polynomial and calling an arbitrary factoring algorithm.

I use the formalised algorithm by Thiemann *et al.* [13, 14], which takes a complex polynomial $p(X)$ and returns some complex number d and a list of pairs (e_i, k_i) such that $p(X) = d \cdot \prod (X - e_i)^{k_i+1}$ and the e_i are distinct. Because we reflected the polynomial, we therefore get $p(X) = d \cdot \prod (1 - e_i X)^{k_i+1}$, which is exactly what we need.

Through the use of the Algebraic Number library by Thiemann *et al.*, this method works out-of-the-box for any rational FPS whose numerator and denominator have rational coefficients, and therefore the overall method works for any linear recurrence with rational coefficients.

3.5 Isabelle Theorems

For the purpose of illustration, I shall print some of the main theorems from the Isabelle formalisation. Some of the notation was adjusted very slightly to make the statements more readable without knowledge of Isabelle, but the statements as printed are still very close to the original ones in Isabelle. In particular, the explicit homomorphisms between $\mathbb{N}$ and $\mathbb{Z}$ or between $K[X]$ and $K[[X]]$ that have to be made explicit in Isabelle are still present.

The following is the statement of the Isabelle theorem about converting a linear homogeneous recurrence to a rational FPS:

lemma
 fixes f :: nat $\Rightarrow$ (α :: field) **and** cs :: α list
 defines N := length $cs - 1$
 assumes $cs \neq []$
 assumes $\forall n \geq m.\ \left(\sum_{k \leq N} cs_k \cdot f\ (n + k)\right) = 0$
 assumes last $cs \neq 0$
 shows Abs_fps f = fps_of_poly (lhr_fps_numerator m cs f) / fps_of_poly (lr_fps_denominator cs)

Here, the α :: *field* stands for an arbitrary type of the type class *field*, which is a field in the algebraic sense. This type class has concrete instances like *rat*, *real*, or *complex*.

The function *Abs_fps* converts between a sequence (a function $\mathbb{N} \longrightarrow \alpha$) and an FPS ($\alpha$ fps). The function *fps_of_poly* is the canonical homomorphism mapping a polynomial to an FPS. Note that all the functions on the right-hand side of the equation in the theorem statement have code equations in Isabelle and are therefore executable.

Furthermore, the theorem on the closed form of a rational FPS is:

lemma
 assumes is_alt_factorization_of *fctrs* q **and** $q \neq 0$
 shows Abs_fps (interp_ratfps_solution (solve_factored_ratfps' p *fctrs*)) = fps_of_poly p / fps_of_poly q

with

definition interpret_ratfps_solution (p, cs) n =
 coeff p n + $(\sum_{(q,c) \leftarrow cs}$ poly q (of_nat n) $\cdot$ c ^ $n)$

Here, *poly* evaluates a polynomial and *of_nat* is the canonical homomorphism from $\mathbb{N}$ into any other semiring. Moreover, *is_alt_factorization_of* states that *fctrs* is a factorization of q into the form $d \prod (1 - e_i X)^{n_i}$ as we have seen before, with *fctrs* being a pair consisting of the d and a list of pairs of the e_i and n_i. The solution that is being computed is a pair of a polynomial p, whose coefficients form the 'correction terms' for recurrences with additional initial values, and a list consisting of pairs of a polynomial $r_i(X)$ and a number c_i such that $r_i(n)\, c_i^n$ is a summand in the solution. The function *interpret_ratfps_solution* 'evaluates' such a solution for a given n.

Since these theorems are somewhat notationally technical, I will not print any more of them here; the correctness theorems for converting inhomogeneous recurrences to FPSs and solving recurrences look very similar to the ones printed here.

However, the following more abstract theorem about the shape of solutions of rational FPSs (and thereby linear recurrences) is probably reasonably readable:

theorem
  fixes $p\ q$ :: complex poly
  assumes coeff q $0 \neq 0$
  defines $q' :=$ reflect_poly q
  obtains $r\ rs$
  where $\forall n.$ fps_nth (fps_of_poly p / fps_of_poly q) n =
    coeff $r\ n + (\sum_{c \mid \text{poly } q'\ c = 0}$ poly $(rs\ c)$ (of_nat n) $\cdot\ c$ ^ n)
  and $\forall z.$ poly $q'\ z = 0 \Longrightarrow$ degree $(rs\ z) \leq$ order $z\ q' - 1$

It states that the solution of a rational FPS is a sum where each root c of the denominator contributes a summand of the form $p(n)c^n$, plus a correction term that vanishes for almost all n. Moreover, the degree of each polynomial is at most the order of the corresponding root minus 1. The variable *rs* in the above Isabelle theorem is the function that associates the polynomial to each root, and the r is a polynomial whose coefficients constitute the 'correction term'.
This also directly implies the key theorem on coefficient asymptotics:

theorem
  fixes $p\ q$ :: complex poly
  assumes square_free_factorization q (b, cs)
  assumes $q \neq 0$ **and** $R > 0$
  assumes $\forall (c, l) {\in} cs.\ \forall x.$ norm $x < 1/R \longrightarrow$ poly $c\ x \neq 0$
  assumes $\forall (c, l) {\in} cs.\ \forall x.\ l > k \wedge$ norm $x = 1/R \longrightarrow$ poly $c\ x \neq 0$
  shows fps_nth (fps_of_poly p / fps_of_poly q) $\in$
    $O(\lambda n.$ of_nat n ^ $k\ \cdot$ of_real R ^ $n)$

The *square_free_factorization* predicate asserts that the given polynomial $q(X)$ has a square-free factorization into some constant b and some list of factors $c_i(X)^{l_i}$ pairs, represented as a list of pairs of the form (c, l).

The Landau symbol $O(\ldots)$ is defined in the usual fashion where $f \in O(g)$ iff there exists some $C \in \mathbb{R}$ such that, for all sufficiently large x, $|f(x)| \leq C|g(x)|$.

3.6 Pretty Printing

One disadvantage of using the Algebraic Number library is that printing irrational algebraic numbers is not straightforward. Without additional setup, evaluating e. g. `sqrt 2` leads to a few lines of fairly illegible output which corresponds to the internal representation of irrational algebraic numbers in the Isabelle library as an integer polynomial with an additional upper and lower bound that uniquely identifies one root of the polynomial, which is the number that is being described.

Thankfully, Thiemann *et al.* also provide some pretty-printing functionality which converts a real-algebraic number into a human-readable string. For algebraic numbers of degree at most 2, this is exactly what one would expect: a combination of rational numbers and square roots. However, for numbers of higher degree, the output is unfortunately still of the form 'k-th root of polynomial p'; e. g. $\sqrt[3]{2}$ is rendered as `root #1 of -2 + x^3, in (1,2)`.

I add to this some more pretty-printing code in order to display rational FPSs and the solutions computed for linear recurrences in a natural, human-readable form.

It should be noted that all of the pretty-printing that Thiemann *et al.* and I do is unverified; however, in both cases, the step from the representation in Isabelle to the human-readable string is very small.

4 Evaluation

We can now evaluate the solver on some examples using Isabelle's `value` and `export_code` commands.

4.1 Fibonacci Numbers

For the Fibonacci case, we invoke `solve_lhr [-1,-1,1] [0,1]`, which – after pretty-printing the complex numbers – returns:

```
Some (0, [(sqrt(1/5), (1/2+sqrt(5/4))),
  ( -sqrt(1/5), (1/2-sqrt(5/4)))])
```

Alternatively, when using the pretty-printer for solutions of recurrences, we get:

```
(sqrt(1/5)) * (1/2+sqrt(5/4)) ^ x +
( -sqrt(1/5)) * (1/2-sqrt(5/4)) ^ x
```

We can also compute the rational FPS of the recurrence using *lhr_fps* directly:

```
-1x / (-1 + x + x^2)
```

4.2 A Higher-Degree Recurrence

Another simple example of higher degree is the sequence $0, 1, 2, 3, 0, 1, 2, 3, \ldots$ with the recurrence $f(n+4) - f(n) = 0$ and the initial values $0, 1, 2, 3$. The output is

```
(-1/2) * (-1) ^ x + (-1/2+1/2i) * (1i) ^ x +
  (-1/2+-1/2i) * (-1i) ^ x + (3/2)
```

or, in a more readable form:

$$f(n) = -\frac{(-1)^n}{2} + \frac{i-1}{2}i^n - \frac{i+1}{2}(-i)^n + \frac{3}{2}$$

4.3 Running Example

Next, let us again consider our running example as an example of an inhomogenous recurrence. This can be solved by evaluating

```
solve_lir [-2, -1, 1] [0, 0] [(1, 1, 2)]
```

which returns

```
(-16/27 + 5/9x + 1/3x^2) * 2 ^ x +
  (16/27) * (-1) ^ x
```

4.4 A pathological Example

Lastly, we look the seemingly innocuous example $5f(n+4) + 4f(n+3) + 3f(n+2) + 2f(n+1) + f(n) = 0$ with arbitrary initial values. The recurrence solver does not terminate for this example and eventually runs out of memory. Factoring the characteristic polynomial itself is no problem, but due to the way algebraic complex numbers are implemented by Thiemann *et al.*, the computations with the roots of this polynomial lead to a significant blow-up in the degrees of the integer polynomials used to represent the real and imaginary parts of complex algebraic numbers, which then need to be factored again.

However, if we only want to know the asymptotics of the solution, things are much more pleasant: The dominant roots of the characteristic polynomial have an absolute value of $1.445046\ldots$, whose reciprocal is ≈ 0.6920196, so we can easily show e. g. $f(n) \in O(0.69202^n)$ automatically.

4.5 Performance Comparison

Evaluating any of these examples directly in Isabelle with the `value` command takes about 1 minute. However, almost all of this is taken up by code generation and compilation. I therefore exported the code for the functions *solve_lhr*, *solve_lir*, and the pretty printing to Haskell using Isabelle's `code_export` command, wrote a minimal wrapper for input/output and to convert Haskell numbers into the exported datatype for complex numbers, and compiled everything with the Glasgow Haskell Compiler.

Comparing the performance of the Isabelle solver to that of systems like Mathematica and Maple does not make much sense because the computation time is completely dominated by the polynomial factorisation, so any attempt to compare the efficiency of the linear recurrence solvers essentially boils down to a mere comparison of the efficiency of the polynomial factorisation algorithms. Nevertheless, Table 1 gives a quick impression of how the verified solver and Mathematica's solver perform on the above examples and some more randomly-generated examples. The performance for certifying a reasonably tight 'Big-O' bound in Isabelle is also given.

The measurements are to be taken with a grain of salt as they were conducted on a shared machine in the computer pool at the Technical University of Munich since this was the only machine on which a Mathematica licence was available. However, repeating the measurements at different times showed that they were fairly stable; they should be good enough to at least give a qualitative comparison of the performance behaviour of the different approaches.

The table shows that Mathematica performs considerably better than the Isabelle solver on the pathological example of degree 5 mentioned before. However, for similarly pathological polynomials of even a slightly higher degree (e. g. 9), Mathematica also fails to terminate within a reasonable amount of time (around 5 minutes). The verified asymptotics certification method works much better than either solver, but its performance also degenerates very quickly as the degrees grow, and its performance also depends on how 'complicated' the fraction b in the $O(n^k b^n)$ is. The reason for this is that the current implementation of Wenda Li's root-counting method uses rational arithmetic and the numerators and denominators can grow very large, depending on the numbers in the input and the degree of the polynomial. This is a known problem with remainder-series-based approaches like Li's.

The performance comparison suggests that solving recurrences externally and somehow certifying results in Isabelle is perhaps not very useful, since the verified solver copes very well with 'simple' recurrences and when one moves to complicated recurrences, the performance of the Mathematica solver also degrades very quickly. Furthermore, since the performance degradation seems to come mainly from polynomials with 'complicated' roots and the irrational arithmetic involved in the resulting computations, it seems likely that any kind of certification in a case like this would also have to involve the same irrational arithmetic since these roots are part of the closed-form solution. It therefore seems doubtful if certification of a closed-form solution makes sense.

However, both the polynomial factorisation by Thiemann *et al.* and the root-counting procedure by Li are under active development and the code developed here will directly profit from any improvements that they make.

There is no analogue to the verified asymptotics certifier in Mathematica or Maple, although it would be fairly easy to write one and it should be very efficient, seeing as systems like Mathematica tend to have very sophisticated algorithms for root isolation. In fact, it may be useful to employ a system like Mathematica to determine the approximate or exact asymptotics of an equation using its root isolation algorithms and then certify it in Isabelle using the verified certifier. However, this has not been implemented yet.

5 Related Work

To my knowledge, this is the first work on the theory and a solver for linear recurrences in a proof assistant. The theory of these recurrences has been known for a long time and is usually taught in undergraduate courses of mathematics. The classic method of applying partial fraction decomposition to the rational FPS that was used in this work can be found in many textbooks [7]. An alternative route is to use results from Analytic Combinatorics [5] for meromorphic functions (of which rational functions are a special case) to obtain the coefficients of the FPS in terms of the complex residues at its poles.

Many different executable solvers for linear recurrences are available, e. g. PURRS [1] or those that come with computer algebra systems like Mathematica and Maple. For the homogeneous part, they also use the approach of factoring the characteristic polynomial (possibly with some preprocessing to decrease the degree of the recurrence). Unlike the Isabelle formalisation, these systems typically support

Table 1. Benchmark results of the solver (the verified one and Mathematica's) and the verified certifier. The first 4 examples are the ones from above; the remaining ones have randomly-generated coefficients ranging from −10 to 10. A time of 0 indicates that the time was below 1 ms; a time of ∞ indicates a timeout after more than 5 minutes. The given asymptotics show an approximation of the actual (irrational) basis.

Example	Degree	Asymptotics	Time (ms)		
			Solver (ver.)	Solver (Math.)	Certifier (ver.)
Fibonacci	2	$O(1.619^n)$	12	20	0
0, 1, 2, 3…	4	$O(1)$	0	12	0
$f(n) + 2f(n-1) = n \cdot 2^n$	3	$O(n \cdot 2^n)$	0	12	0
Running example	5	$O(n^2 \cdot 2^n)$	7	368	0
Pathological	5	$O(0.69202^n)$	∞	4600	0
Random	3	$O(2.331^n)$	1200	970	0
Random	9	$O(1.1552^n)$	∞	∞	260
Random	11	$O(8.876^n)$	∞	∞	5100
Random	14	$O(1.1985^n)$	∞	∞	238000

more complicated inhomogeneous parts. This typically requires finding a closed form for some symbolic sum or 'guessing' a solution. Furthermore, they also allow solving systems of recurrences. Both of this is beyond the scope of this work.

6 Conclusion

I formalised the basic theory of and an executable solver for linear recurrences with constant coefficients, both homogeneous ones and inhomogeneous ones where the inhomogeneous term is of polynomial-exponential form. An executable solver for these recurrences and a more efficient certifier for the 'Big-O' asymptotics of their solutions is also provided. This development makes use of many different components:

- Executable polynomial factorisation and algebraic numbers (Thiemann *et al.* [13, 14])
- Square-free polynomial factorisation (Thiemann *et al.* [15, 16])
- Formal power series (Chaieb [3])
- Stirling Numbers (Isabelle library)
- Counting complex roots (Li [9, 10])
- Eulerian polynomials (Eberl)
- Partial Fraction Decomposition (Eberl)
- Executable rational FPSs (Eberl)

The last three of these were motivated by this very application of solving linear recurrences and the modularity allows us to e. g. easily improve the Partial Fraction Decomposition algorithm in the future, or to directly benefit from any performance improvements made by Thiemann *et al.* without any changes to the recurrence solver.

Acknowledgments

This work was supported by Deutsche Forschungsgemeinschaft under Grants DFG RTG 1480 (PUMA) and DFG Koselleck NI 491/16-1.

Lastly, I would like to thank René Thiemann and Akihisa Yamada for their help with their Algebraic Numbers library and Wenda Li for implementing the circular case in his root-counting algorithm upon my suggestion. I also thank the anonymous reviewers for their advice.

References

[1] R. Bagnara, A. Zaccagnini, and T. Zolo. 2003. *The Automatic Solution of Recurrence Relations. I. Linear Recurrences of Finite Order with Constant Coefficients.* Quaderno 334. Dipartimento di Matematica, Università di Parma, Italy. Available at http://www.cs.unipr.it/Publications/.

[2] Lukas Bulwahn. 2012. *The New Quickcheck for Isabelle.* Springer Berlin Heidelberg, Berlin, Heidelberg, 92–108. https://doi.org/10.1007/978-3-642-35308-6_10

[3] Amine Chaieb. 2011. Formal Power Series. *Journal of Automated Reasoning* 47, 3 (01 Oct 2011), 291–318. https://doi.org/10.1007/s10817-010-9195-9

[4] Manuel Eberl. 2017. Linear Recurrences. *Archive of Formal Proofs* (Oct. 2017). http://isa-afp.org/entries/Linear_Recurrences.html, Formal proof development.

[5] Philippe Flajolet and Robert Sedgewick. 2009. *Analytic Combinatorics* (1 ed.). Cambridge University Press, New York, NY, USA.

[6] Philippe Flajolet, Paul Zimmermann, and Bruno Salvy. 1989. *Lambda-Upsilon-Omega: The 1989 cookbook.* Research Report RR-1073. INRIA. https://hal.inria.fr/inria-00075486

[7] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. 1994. *Concrete Mathematics: A Foundation for Computer Science* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] Ondřej Kunčar. 2016. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic.* Ph.D. Dissertation. https://www21.in.tum.de/~kuncar/documents/kuncar-phdthesis.pdf

[9] Wenda Li. 2017. Count the Number of Complex Roots. *Archive of Formal Proofs* (Oct. 2017). http://isa-afp.org/entries/Count_Complex_Roots.html, Formal proof development.

[10] Wenda Li and Lawrence C. Paulson. 2018. Evaluating Winding Numbers and Counting Complex Roots through Cauchy Indices in Isabelle/HOL. *CoRR* abs/1804.03922 (2018). http://arxiv.org/abs/1804.03922

[11] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs.

In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. ACM, New York, NY, USA, 496–512. https://doi.org/10.1145/3192366.3192394

[12] Joël Ouaknine and James Worrell. 2015. On Linear Recurrence Sequences and Loop Termination. *ACM SIGLOG News* 2, 2 (April 2015), 4–13. https://doi.org/10.1145/2766189.2766191

[13] René Thiemann and Akihisa Yamada. 2015. Algebraic Numbers in Isabelle/HOL. *Archive of Formal Proofs* (Dec. 2015). http://isa-afp.org/entries/Algebraic_Numbers.html Formal proof development.

[14] René Thiemann and Akihisa Yamada. 2016. *Algebraic numbers in Isabelle/HOL*. Springer International Publishing, Cham, 391–408. https://doi.org/10.1007/978-3-319-43144-4_24

[15] René Thiemann and Akihisa Yamada. 2016. Formalizing Jordan Normal Forms in Isabelle/HOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2016)*. ACM, New York, NY, USA, 88–99. https://doi.org/10.1145/2854065.2854073

[16] René Thiemann and Akihisa Yamada. 2016. Polynomial Factorization. *Archive of Formal Proofs* (Jan. 2016). http://isa-afp.org/entries/Polynomial_Factorization.html, Formal proof development.

[17] Bohua Zhan and Maximilian P. L. Haslbeck. 2018. Verifying Asymptotic Time Complexity of Imperative Programs in Isabelle. *CoRR* abs/1802.01336 (2018). http://arxiv.org/abs/1802.01336